OPTIMIZATION-BASED APPROXIMATE DYNAMIC PROGRAMMING

A Dissertation Presented

by

MAREK PETRIK

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 2010

Department of Computer Science

OPTIMIZATION-BASED APPROXIMATE DYNAMIC PROGRAMMING

A Dissertation Presented

by

MAREK PETRIK

Approved as to style and content by:

Shlomo Zilberstein, Chair

Andrew Barto, Member

Sridhar Mahadevan, Member

Ana Muriel, Member

Ronald Parr, Member

Andrew Barto, Department Chair
Department of Computer Science

To my parents Fedor and Mariana

ACKNOWLEDGMENTS

I want to thank the people who made my stay at UMass not only productive, but also very enjoyable. I am grateful to my advisor, Shlomo Zilberstein, for guiding and supporting me throughout the completion of this work. Shlomo's thoughtful advice and probing questions greatly influenced both my thinking and research. His advice was essential not only in forming and refining many of the ideas described in this work, but also in assuring that I become a productive member of the research community. I hope that, one day, I will be able to become an advisor who is just as helpful and dedicated as he is.

The members of my dissertation committee were indispensable in forming and steering the topic of this dissertation. The class I took with Andrew Barto motivated me to probe the foundations of reinforcement learning, which became one of the foundations of this thesis. Sridhar Mahadevan's exciting work on representation discovery led me to deepen my understanding and appreciate better approximate dynamic programming. I really appreciate the detailed comments and encouragement that Ron Parr provided on my research and thesis drafts. Ana Muriel helped me to better understand the connections between my research and applications in operations research. Coauthoring papers with Jeff Johns, Bruno Scherrer, and Gavin Taylor was a very stimulating and learning experience. My research was also influenced by interactions with many other researches. The conversations with Raghav Aras, Warren Powell, Scott Sanner, and Csaba Szepesvari were especially illuminating. This work was also supported by generous funding from the Air Force Office of Scientific Research.

Conversations with my lab mate Hala Mostafa made the long hours in the lab much more enjoyable. While our conversations often did not involve research, those that did, motivated me to think deeper about the foundations of my work. I also found sharing ideas with my fellow grad students Martin Allen, Chris Amato, Alan Carlin, Phil Kirlin, Akshat Kumar, Sven Seuken, Siddharth Srivastava, and Feng Wu helpful in understanding the broader research topics. My free time at UMass kept me sane thanks to many great friends that I found here.

Finally and most importantly, I want thank my family. They were supportive and helpful throughout the long years of my education. My mom's loving kindness and my dad's intense fascination with the world were especially important in forming my interests and work habits. My wife Jana has been an incredible source of support and motivation in both research and private life; her companionship made it all worthwhile. It was a great journey.

ABSTRACT

OPTIMIZATION-BASED APPROXIMATE DYNAMIC PROGRAMMING

SEPTEMBER 2010

MAREK PETRIK

Mgr., UNIVERZITA KOMENSKEHO, BRATISLAVA, SLOVAKIA

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Shlomo Zilberstein

Reinforcement learning algorithms hold promise in many complex domains, such as resource management and planning under uncertainty. Most reinforcement learning algorithms are iterative — they successively approximate the solution based on a set of samples and features. Although these iterative algorithms can achieve impressive results in some domains, they are not sufficiently reliable for wide applicability; they often require extensive parameter tweaking to work well and provide only weak guarantees of solution quality.

Some of the most interesting reinforcement learning algorithms are based on approximate dynamic programming (ADP). ADP, also known as value function approximation, approximates the value of being in each state. This thesis presents new reliable algorithms for ADP that use optimization instead of iterative improvement. Because these *optimization–based* algorithms explicitly seek solutions with favorable properties, they are easy to analyze, offer much stronger guarantees than iterative algorithms, and have few or no parameters to tweak. In particular, we improve on approximate linear programming — an existing method — and derive approximate bilinear programming — a new robust approximate method.

The strong guarantees of optimization–based algorithms not only increase confidence in the solution quality, but also make it easier to combine the algorithms with other ADP components. The other components of ADP are samples and features used to approximate the value function. Relying on the simplified analysis of optimization–based methods, we derive new bounds on the error due to missing samples. These bounds are simpler, tighter, and more practical than the existing bounds for iterative algorithms and can be used to evaluate solution quality in practical settings. Finally, we propose homotopy methods that use the sampling bounds to automatically select good approximation features for optimization–based algorithms. Automatic feature selection significantly increases the flexibility and applicability of the proposed ADP methods.

The methods presented in this thesis can potentially be used in many practical applications in artificial intelligence, operations research, and engineering. Our experimental results show that optimization–based methods may perform well on resource-management problems and standard benchmark problems and therefore represent an attractive alternative to traditional iterative methods.

CONTENTS

Page

ACKNOWLEDGMENTS . . . . **v**

ABSTRACT . . . . **vi**

LIST OF FIGURES . . . . **xii**

CHAPTER

1. INTRODUCTION **1**

1.1 Planning Models 2
1.2 Challenges and Contributions 3
1.3 Outline 6

PART I: FORMULATIONS

2. FRAMEWORK: APPROXIMATE DYNAMIC PROGRAMMING **9**

2.1 Framework and Notation 9
2.2 Model: Markov Decision Process 10
2.3 Value Functions and Policies 12
2.4 Approximately Solving Markov Decision Processes 16
2.5 Approximation Error: Online and Offline 22
2.6 Contributions 24

3. ITERATIVE VALUE FUNCTION APPROXIMATION **25**

3.1 Basic Algorithms 25
3.2 Bounds on Approximation Error 28
3.3 Monotonous Approximation: Achieving Convergence 30
3.4 Contributions 31

4. APPROXIMATE LINEAR PROGRAMMING: TRACTABLE BUT LOOSE APPROXIMATION 33

4.1 Formulation 33

4.2 Sample-based Formulation 36

4.3 Offline Error Bounds 38

4.4 Practical Performance and Lower Bounds 39

4.5 Expanding Constraints 42

4.6 Relaxing Constraints 45

4.7 Empirical Evaluation 49

4.8 Discussion 50

4.9 Contributions 51

5. APPROXIMATE BILINEAR PROGRAMMING: TIGHT APPROXIMATION 52

5.1 Bilinear Program Formulations 52

5.2 Sampling Guarantees 59

5.3 Solving Bilinear Programs 60

5.4 Discussion and Related ADP Methods 61

5.5 Empirical Evaluation 65

5.6 Contributions 67

PART II: ALGORITHMS

6. HOMOTOPY CONTINUATION METHOD FOR APPROXIMATE LINEAR PROGRAMS 69

6.1 Homotopy Algorithm 69

6.2 Penalty-based Homotopy Algorithm 73

6.3 Efficient Implementation 76

6.4 Empirical Evaluation 78

6.5 Discussion and Related Work 79

6.6 Contributions 81

7. SOLVING APPROXIMATE BILINEAR PROGRAMS 82

7.1 Solution Approaches 82

7.2 General Mixed Integer Linear Program Formulation 83

7.3 ABP-Specific Mixed Integer Linear Program Formulation 85

7.4 Homotopy Methods 87

7.5 Contributions 89

8. SOLVING SMALL-DIMENSIONAL BILINEAR PROGRAMS 90

8.1 Bilinear Program Formulations 90

8.2 Dimensionality Reduction 92

8.3 Successive Approximation Algorithm 95

8.4 Online Error Bound 99

8.5 Advanced Pivot Point Selection 101

8.6 Offline Bound 106

8.7 Contributions 107

PART III: SAMPLING, FEATURE SELECTION, AND SEARCH

9. SAMPLING BOUNDS 109

9.1 Sampling In Value Function Approximation 110

9.2 State Selection Error Bounds 111

9.3 Uniform Sampling Behavior 116

9.4 Transition Estimation Error 117

9.5 Implementation of the State Selection Bounds 120

9.6 Discussion and Related Work 122

9.7 Empirical Evaluation 124

9.8 Contributions 128

10.FEATURE SELECTION 129

10.1 Feature Considerations 129

10.2 Piecewise Linear Features 130

10.3 Selecting Features 132

10.4 Related Work 135

10.5 Empirical Evaluation 137

10.6 Contributions 138

11.HEURISTIC SEARCH 140

11.1 Introduction 140

11.2 Search Framework 144

11.3 Learning Heuristic Functions 147

11.4 Feature Combination as a Linear Program 156

11.5 Approximation Bounds 161
11.6 Empirical Evaluation 165
11.7 Contributions 167

12.CONCLUSION 168

PART: APPENDICES

APPENDICES

A. NOTATION 171
B. PROBLEM DESCRIPTIONS 173
C. PROOFS 178

BIBLIOGRAPHY 238

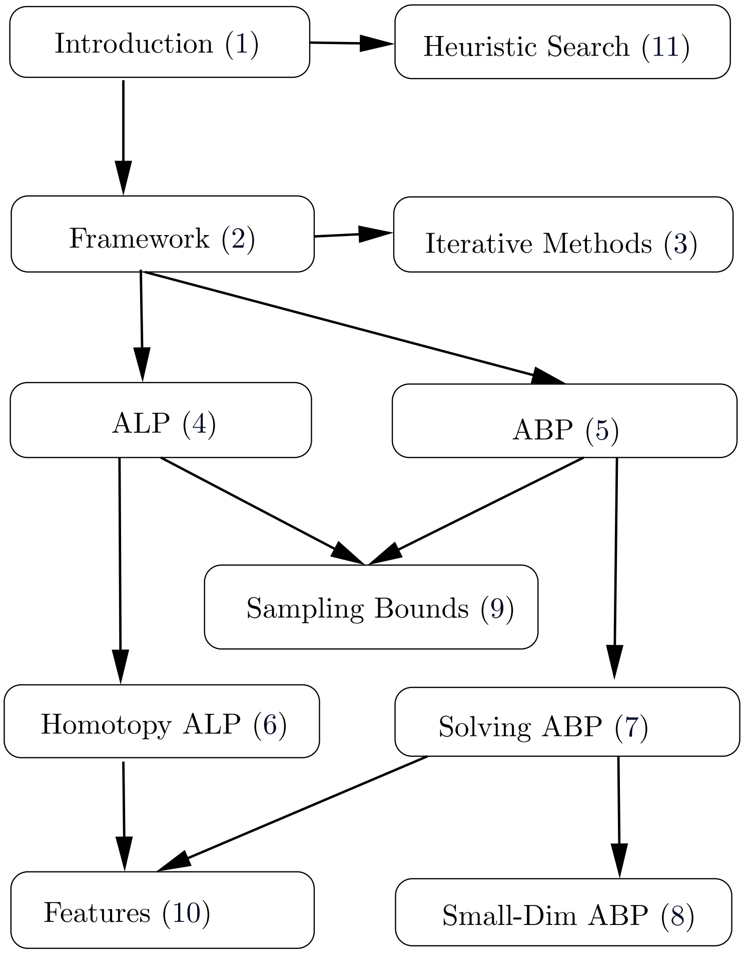

Figure 1. Chapter Dependence

CHAPTER 1

INTRODUCTION

Planning is the process of creating a sequence of actions that achieve some desired goals. Typical planning problems are characterized by a structured state space, a set of possible actions, a description of the effects of each action, and an objective function. This thesis treats planning as an optimization problem, seeking to find plans that minimize the cost of reaching the goals or some other performance measure.

Automatic planning in large domains is one of the hallmarks of intelligence and is, as a result, a core research area of artificial intelligence. Although the development of truly general artificial intelligence is decades — perhaps centuries — away, problems in many practical domains can benefit tremendously from automatic planning. Being able to adaptively plan in an uncertain environment can significantly reduce costs, improve efficiency, and relieve human operators from many mundane tasks. The two planning applications below illustrate the utility of automated planning and the challenges involved.

Blood inventory management A blood bank aggregates a supply of blood and keeps an inventory to satisfy hospital demands. The hospital demands are stochastic and hard to predict precisely. In addition, blood ages when it is stored and cannot be kept longer than a few weeks. The decision maker must decide on blood-type substitutions that minimize the chance of future shortage. Because there is no precise model of blood demand, the solution must be based on historical data. Even with the available historical data, calculating the optimal blood-type substitution is a large stochastic problem. Figure 1.1 illustrates the blood inventory management problem and Section B.2 describes the formulation in more detail.

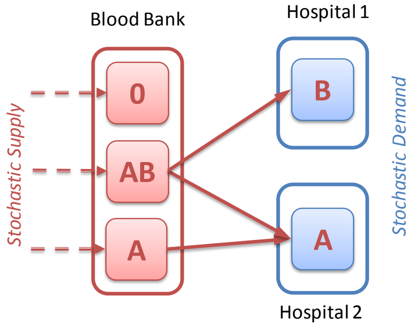

Figure 1.1. Blood Inventory Management

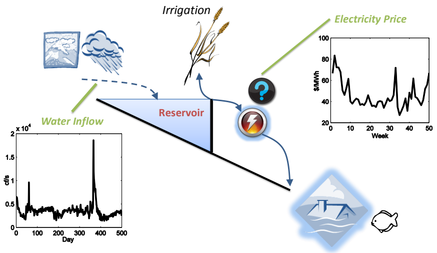

Figure 1.2. Reservoir Management

Water reservoir management An operator needs to decide how much and when to discharge water from a river dam in order to maximize energy production, while satisfying irrigation and flood control requirements. The challenges in this domain are in some sense complementary to blood inventory management with fewer decision options but greater uncertainty in weather and energy prices. Figure 1.2 illustrates the reservoir management problem and Section B.3 describes the formulation in more detail.

Many practical planning problems are solved using *domain-specific* methods. This entails building specialized models, analyzing their properties, and developing specialized algorithms. For example, blood inventory and reservoir management could be solved using the standard theory of inventory management (Axsater, 2006). The drawback of specialized methods is their limited applicability. Applying them requires significant human effort and specialized domain knowledge. In addition, the domain can often change during the lifetime of the planning system.

Domain-specific methods may also be inapplicable if the domain does not clearly fit into an existing category. For example, because of the compatibility among blood types, blood inventory management does not fit well the standard inventory control framework. In reservoir management, the domain-specific methods also do not treat uncertainty satisfactorily, neither do they work easily with historical data (Forsund, 2009).

This thesis focuses on general planning methods — which are easy to apply to a variety of settings — as an alternative to domain–specific ones. Having general methods that can reliably solve a large variety of problems in many domains would enable widespread application of planning techniques. The pinnacle of automatic planning, therefore, is to develop algorithms that work reliably in many settings.

1.1 Planning Models

The purpose of planning models is to capture prior domain knowledge, which is crucial in developing appropriate algorithms. The models need to capture the simplifying properties of given domains and enable more general application of the algorithms.

A planning model must balance domain specificity with generality. Very domain-specific models, such as the traveling salesman problem, may be often easy to study but have limited applicability. In comparison, very general models, such as mixed integer linear programs, are harder to study and their solution methods are often inefficient.

Aside from generality, the suitability of a planning model depends on the specific properties of the planning domain. For example, many planning problems in engineering can be modeled using partial differential equations (Langtangen, 2003). Such models can capture faithfully continuous physical processes, such as rotating a satellite in the orbit. These models also lead to general, yet well-performing methods, which work impressively well in complex domains. It is, however, hard to model and address stochasticity using partial differential equations. Partial differential equations also cannot be easily used to model discontinuous processes with discrete states.

Markov decision process (MDP) — the model used in this thesis — is another common general planning model. The MDP is a very simple model; it models only states, actions, transitions, and associated rewards. MDPs focus on capturing the uncertainty in domains with complex state transitions. This thesis is concerned with maximizing the discounted infinite-horizon sum of the rewards. That means that the importance of future outcomes is discounted, but not irrelevant. Infinite-horizon discounted objectives are most useful in long-term maintenance domains, such as inventory management.

MDPs have been used successfully to model a wide variety of problems in computer science — such as object recognition, sensor management, and robotics — and in operations research and engineering — such as energy management for a hybrid electric vehicle, or generic resource management (Powell, 2007a). These problems must be solved approximately, because of their large scale and imprecise models. It is remarkable that all of these diverse domains can be modeled, analyzed, and solved using a single framework.

The simplicity and generality of MDP models has several advantages. First, it is easy to learn these models based on historical data, which is often abundant. This alleviates the need for manual parsing of the data. Second, it is easy to add assumptions to the MDP model to make it more suitable for a specific domain (Boutilier, Dearden, & Goldszmidt, 1995; Feng, Hansen, & Zilberstein, 2003; Feng & Zilberstein, 2004; Guestrin, Koller, Parr, & Venkataraman, 2003; Hansen & Zilberstein, 2001; Hoey & Poupart, 2005; Mahadevan, 2005b; Patrascu, Poupart, Schuurmans, Boutilier, & Guestrin, 2002; Poupart, Boutilier, Patrascu, & Schuurmans, 2002). One notable example is the model used in classical planning. Models of classical planning are generally specializations of MDPs that add structure to the state space based on logical expressions. These models are discussed in more detain in Chapter 11.

1.2 Challenges and Contributions

Although MDPs are easy to formulate, they are often very hard to solve. Solving large MDPs is a computationally challenging problem addressed widely in the artificial intelligence — particularly reinforcement learning — operations research, and engineering literature. It is widely accepted that large MDPs can only be solved approximately. Approximate solutions may be based on samples of the domain, rather than the full descriptions.

An MDP consists of states $\mathcal{S}$ and actions $\mathcal{A}$. The solution of an MDP is a *policy* $\pi : \mathcal{S} \to \mathcal{A}$, or a decision rule, which assigns an action to each state. A related solution concept is the *value function* $v : \mathcal{S} \to \mathbb{R}$, which represents the expected value of being in every state. A value function v can be easily used to construct a greedy policy $\pi(v)$. It is useful to study value functions, because they are easier to analyze than policies. For any policy π, the *policy loss* is the difference between return of the policy and the return of an optimal policy. Because it is not feasible to compute an optimal policy, the goal of this work is to compute a policy with a small policy loss $l(\pi) \in \mathbb{R}$.

Approximate methods for solving MDPs can be divided into two broad categories: 1) *policy search*, which explores a restricted space of all policies, 2) *approximate dynamic programming*, which searches a restricted space of value functions. While all of these methods have achieved impressive results in many domains, they have significant limitations, some of which are addressed in this thesis.

Policy search methods rely on local search in a restricted policy space. The policy may be represented, for example, as a finite–state controller (Stanley & Miikkulainen, 2004) or as a greedy policy with respect to an approximate value function (Szita & Lorincz, 2006). Policy search methods have achieved impressive results in such domains as Tetris (Szita & Lorincz, 2006) and helicopter control (Abbeel, Ganapathi, & Ng, 2006). However, they are notoriously hard to analyze. We are not aware of any theoretical guarantees regarding the quality of the solution. This thesis does not address policy search methods in any detail.

Approximate dynamic programming (ADP) — also known as value function approximation — is based on computing value functions as an intermediate step to computing polices. Most ADP methods iteratively approximate the value function (Bertsekas & Ioffe, 1997; Powell, 2007a; Sutton & Barto, 1998). Traditionally, ADP methods are defined *procedurally*; they are based on precise methods for solving MDPs with an approximation added. For example, approximate policy iteration — an approximate dynamic programming method — is a variant of policy iteration. The procedural approach leads to simple algorithms that may often perform well. However, these algorithms have several theoretical problems that make them impractical in many settings.

Although procedural (or iterative) ADP methods have been extensively studied and analyzed, their is still limited understanding of their properties. They do not converge and therefore do not provide finite-time guarantees on the size of the policy loss. As a result, procedural ADP methods typically require significant domain knowledge to work (Powell, 2007a); for example they are sensitive to the approximation features (Mahadevan, 2009). The methods are also sensitive to the distribution of the samples used to calculate the solution (Lagoudakis & Parr, 2003) and many other problem parameters. Because the sensitivity is hard to quantify, applying the existing methods in unknown domains can be very challenging.

This thesis proposes and studies a new *optimization–based approach* to approximate dynamic programming as an alternative to traditional iterative methods. Unlike procedural ADP methods, optimization–based ADP methods are defined declaratively. In the *declarative* approach to ADP, we first explicitly state the desirable solution properties and then develop algorithms that can compute such solution. This leads to somewhat more involved algorithms, but ones that are much easier to analyze. Because these optimization techniques are defined in terms of specific properties of value functions, their results are easy to analyze and they provide strong guarantees. In addition, the formulations essentially decouple the actual algorithm used from the objective, which increases the flexibility of the framework.

The objective of optimization–based ADP is to compute a value function v that leads to a policy $\pi(v)$ with a *small policy loss* $l(\pi(v))$. Unfortunately, the policy loss $l \circ \pi : (\mathcal{S} \to \mathbb{R}) \to \mathbb{R}$ as a function of the value function lacks structure and cannot be efficiently computed without simulation. We, therefore, derive *upper bounds* $f : (\mathcal{S} \to \mathbb{R}) \to \mathbb{R}$ such that $f \geq l$ on the policy loss that are easy to evaluate and optimize, as depicted in Figure 1.3. We

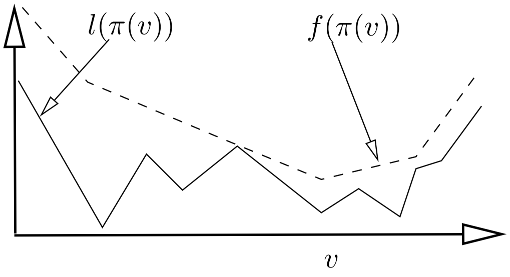

Figure 1.3. Optimization Framework

then develop methods that compute $\arg\min_v f$ as a proxy for $\arg\min_v l$ and analyze the error due to the proxy optimization.

Approximate linear programming (ALP), which can be classified as an optimization–based approach to ADP, has been proposed and studied previously (Schweitzer & Seidmann, 1985; de Farias, 2002). ALP uses a linear program to compute the approximate value function in a particular vector space (de Farias, 2002). ALP has been previously used in a wide variety of settings (Adelman, 2004; de Farias & van Roy, 2004; Guestrin et al., 2003). ALP has better theoretical properties than iterative approximate dynamic programming and policy search. However, the L_1 norm must be properly weighted to guarantee a small policy loss, and there is no *reliable* method for selecting appropriate weights (de Farias, 2002). Among other contributions, this thesis proposes modifications of ALP that improve its performance and methods that simultaneously optimize the weights with the value function.

Value function approximation — or approximate dynamic programming — is only one of many components that are needed to solve large MDPs. Other important components are the domain samples and features — or representation — used to approximate the value function. The features represent the prior knowledge. It is desirable to develop methods that are less sensitive to the choice of the features or are able to discover them automatically. It is easier to specify good features for optimization–based algorithms than for iterative value function optimization. The guarantees on the solution quality of optimization–based methods can be used to guide feature selection for given domain samples.

The principal contribution of this thesis is the formulation and study of optimization–based methods for approximate dynamic programming. The thesis also investigates how these methods can be used for representation (feature) selection. The contributions are organized as:

1. New and improved optimization–based methods for approximate dynamic programming. *[Part I]*
   (a) New bounds on the quality of an approximate value function. *[Chapter 2]*
   (b) Lower bounds on the performance of iterative approximate dynamic programming. *[Chapter 3]*
   (c) Improved formulations of approximate linear programs. *[Chapter 4]*
   (d) Tight bilinear formulation of value function approximation. *[Chapter 5]*
2. Algorithms for solving optimization–based dynamic programs. *[Part II]*
   (a) Homotopy continuation methods for solving optimization–based formulation. *[Chapter 6]*

   (b) Approximate algorithms for optimization–based formulations. *[Chapter 7]*
   (c) Methods for more efficiently solving some classes of bilinear programs involved in value function approximation. *[Chapter 8]*
3. Methods for selecting representation. *[Part III]*
   (a) Sampling bounds for optimization–based methods. *[Chapter 9]*
   (b) Representation selection based on sampling bounds and the homotopy methods. *[Chapter 10]*
4. Connections between value function approximation and classical planning. *[Chapter 11]*

1.3 Outline

The thesis is divided into three main parts. Part I is concerned with quality measures of approximate value functions and how to formulate them as optimization problems. The formulations are linear and nonlinear mathematical programs. In particular, Chapter 2 describes the Markov decision process framework, notation, and terms. It also defines the crucial concepts with respect to the quality of approximate value functions. Chapter 3 overviews the classic iterative methods for value function approximation and shows that they intrinsically cannot offer the same guarantees as optimization–based methods. Chapter 4 overviews approximate linear programming, an existing optimization–based method, and identifies key problems with the formulation. The chapter also proposes and analyzes methods for addressing these issues. Finally, Chapter 5 proposes approximate bilinear programs — a new formulation that offers the tightest possible guarantees in some instances.

The second part, Part II, then describes methods that can be used to solve the mathematical programs. These methods build on the specific properties of the formulations for increased efficiency. In particular, Chapter 6 proposes a homotopy continuation method for solving approximate linear programs. This method can be used to solve very large approximate linear programs and can also be used to select the appropriate expressivity of the approximation features. Chapter 7 overviews the difficulties with solving approximate bilinear programs using existing methods and proposes new approximate algorithms. Because bilinear programs are inherently hard to solve, Chapter 8 describes methods that can be used to approximately solve some classes of approximate bilinear programs.

Finally, Part III describes how the proposed optimization–based techniques can be used to select and discover features in approximate dynamic programming. In particular, Chapter 9 shows how the structure of a domain together with the structure of the samples can be used to bound the performance-loss due to insufficient domain samples. Approximate dynamic programming, and the optimization–based methods in particular, can overfit the solution if given rich sets of features. Because flexible solution methods must be able to handle rich representation spaces, Chapter 10 proposes methods that can be used to automatically select appropriate feature composition for a given set of samples. Finally, Chapter 11 describes connections between approximate dynamic programming and learning heuristic functions in classical search.

A comprehensive list of all symbols used in the thesis can be found in Appendix A. Appendix B describes the benchmark problems used throughout the thesis. Most of the proofs and technical properties are provided in Appendix C. Figure 1 shows the significant dependencies between chapters and the suggested reading order. Finally, most chapters conclude with a short summary of the contributions presented in the chapter.

PART I

FORMULATIONS

CHAPTER 2

FRAMEWORK: APPROXIMATE DYNAMIC PROGRAMMING

This chapter introduces the planning framework used in this thesis. This framework is a simplification of the general reinforcement learning framework, which often assumes that the process and its model are unknown and are only revealed through acting. We, on the other hand, generally assume that samples of behavior histories are available in advance. Therefore, while some reinforcement learning algorithms interleave optimization and acting, we treat them as two distinct phases.

Domain samples can either come from *historical data* or from a *generative model.* A generative model can simulate the behavior of the states in a domain, but does not necessarily describe it analytically. Such generative models are often available in industrial applications.

Treating the sample generation and optimization as distinct processes simplifies the analysis significantly. There is no need to tradeoff exploration for exploitation. While there is still some cost associated with gathering samples, it can be analyzed independently from runtime rewards.

2.1 Framework and Notation

The thesis relies on an analysis of linear vector spaces. All vectors are column vectors by default. Unless specified otherwise, we generally assume that the vector spaces are finite-dimensional. We use $\mathbf{1}$ and $\mathbf{0}$ to denote vectors of all ones or zeros respectively of an appropriate size. The vector $\mathbf{1}_i$ denotes the vector of all zeros except i-th element, which is one. The operator $\mathbf{I}$ represents an identity matrix of an appropriate size.

Definition 2.1. Assume that $x \in \mathbb{R}^n$ is a vector. A polynomial L_p norm, a weighted L_1 norm, an L_∞ norm, and a span seminorm $\| \cdot \|_s$ are defined as follows respectively:

$$\|x\|_p^p = \sum_i |x(i)|^p \qquad \|x\|_{1,c} = \sum_i c(i)|x(i)|$$

$$\|x\|_\infty = \max_i |x(i)| \qquad \|x\|_s = \max_i x(i) - \min_i x(i).$$

Span of a vector defines a seminorm, which satisfies all the properties of a norm except that $\|x\|_s = 0$ does not imply $x = \mathbf{0}$. We also use:

$$[x]_+ = \max\{x, \mathbf{0}\} \qquad [x]_- = \min\{x, \mathbf{0}\}.$$

The following common properties of norms are used in the thesis.

Proposition 2.2. *Let $\|\cdot\|$ be an arbitrary norm. Then:*

$$\begin{aligned}\|cx\| &= |c|\|x\| \\ \|x+y\| &\le \|x\| + \|y\| \\ |x^{\mathsf{T}} y| &\le \|x\|_p \|y\|_{1-1/p} \\ |x^{\mathsf{T}} y| &\le \|x\|_1 \|y\|_\infty\end{aligned}$$

2.2 Model: Markov Decision Process

The particular planning model is the Markov decision process (MDP). Markov decision processes come in many flavors based on the function that is optimized. Our focus is on the infinite-horizon discounted MDPs, which are defined as follows.

Definition 2.3 (e.g. (Puterman, 2005))**.** A *Markov Decision Process* is a tuple $(\mathcal{S}, \mathcal{A}, P, r, \alpha)$. Here, $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ is the transition function ($P(s,a,s')$ is the probability of transiting to state s' from state s given action a), and $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}_+$ is a reward function. The initial distribution is: $\alpha : \mathcal{S} \mapsto [0,1]$, such that $\sum_{s\in\mathcal{S}} \alpha(s) = 1$.

The goal of the work is to find a sequence of actions that maximizes γ-discounted cumulative sum of the rewards, also called the *return*. A solution of a Markov decision process is a policy, which is defined as follows.

Definition 2.4. A deterministic stationary *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$ assigns an action to each state of the Markov decision process. A stochastic policy *policy* $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$. The set of all stochastic stationary policies is denoted as Π.

General non-stationary policies may take different actions in the same state in different time-steps. We limit our treatment to stationary policies, since for infinite-horizon MDPs there exists an optimal *stationary* and *deterministic* policy. We also consider stochastic policies because they are more convenient to use in some settings that we consider.

We assume that there is an implicit ordering on the states and actions, and thus it is possible to represent any function on $\mathcal{S}$ and $\mathcal{A}$ as either a function, or as a vector according to the ordering. Linear operators on these functions then correspond to matrix operations.

Given a policy π, the MDP turns into a Markov reward process; that is a Markov chain with rewards associated with every transition. Its transition matrix is denoted as P_π and represents the probability of transiting from the state defined by the row to the state defined by the column. For any to states $s, s' \in \mathcal{S}$:

$$P_\pi(s,s') = \sum_{a\in\mathcal{A}} P(s,s,s')\pi(s,a).$$

The reward for the process is denoted as r_π, and is defined as:

$$r_\pi(s) = \sum_{a\in\mathcal{A}} r(s,a)\pi(s,a).$$

To simplify the notation, we use P_a to denote the stochastic matrix that represents the transition probabilities, given a policy π, such that $\pi(s) = a$ for all states $s \in \mathcal{S}$. We also use $P(s,a)$ to denote the vector of $[P(s,a,s_1), P(s,a,s_2)\ldots]$.

The optimization objective, or a *return*, for a policy π is expressed as:

$$\mathbf{E}_{s_0 \sim \alpha} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = \pi(s_0), \ldots, a_t = \pi(s_t) \right],$$

where the expectation is over the transition probabilities. Using the matrix notation defined above, the problem of fining an optimal policy can be restated as follows:

$$\pi^* \in \arg\max_{\pi \in \Pi} \sum_{t=0}^{\infty} \alpha^\mathsf{T} (\gamma P_\pi)^t r_\pi = \arg\max_{\pi \in \Pi} \alpha^\mathsf{T} \left(\mathbf{I} - \gamma P_\pi\right)^{-1} r_\pi.$$

The equality above follows directly from summing the geometric sequence. The optimal policies are often calculated using a value function, defined below.

A *value function* $v : \mathcal{S} \to \mathbb{R}$ assigns a real value to each state. This may be an arbitrary function, but it is meant to capture the utility of being in a state. A value function for a policy π is defined as follows:

$$v_\pi = \sum_{t=0}^{\infty} (\gamma P_\pi)^t r_\pi = \left(\mathbf{I} - \gamma P_\pi\right)^{-1} r_\pi$$

The return for a policy π with a value function v_π can easily be shown to be $\alpha^\mathsf{T} v_\pi$. The optimal value function v^* is the value function of the optimal policy π^*. The formal definition of an optimal policy and value function follows.

Definition 2.5. A policy π^* with the value function v^* is *optimal* if for every policy π:

$$\alpha^\mathsf{T} v^* = \alpha^\mathsf{T} (\mathbf{I} - \gamma P_{\pi^*})^{-1} r_{\pi^*} \geq \alpha^\mathsf{T} (\mathbf{I} - \gamma P_\pi)^{-1} r_\pi.$$

The basic properties of value functions and policies are described in Section 2.3.

The action-value function $q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ for a policy π, also known as Q-function, is defined similarly to value function:

$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s \in \mathcal{S}} v_\pi(s).$$

An optimal action-value function q^* is the action-value function of the optimal policy. We use $q(a)$ to denote a subset of elements of q that correspond to action a; that is $q(\cdot, a)$.

In addition, a policy π induces a *state visitation frequency* $u_\pi : \mathcal{S} \to \mathbb{R}$, defined as follows:

$$u_\pi = \left(\mathbf{I} - \gamma P_\pi^\mathsf{T}\right)^{-1} \alpha.$$

The return of a policy depends on the state-action visitation frequencies and it is easy to show that $\alpha^\mathsf{T} v_\pi = r^\mathsf{T} u_\pi$. The optimal state-action visitation frequency is u_{π^*}. *State-action visitation frequency* $u : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are defined for all states and actions. Notice the missing subscript and that the definition is for $\mathcal{S} \times \mathcal{A}$ not just states. We use u_a to denote the part of u that corresponds to action $a \in \mathcal{A}$. State-action visitation frequencies must satisfy:

$$\sum_{a \in \mathcal{A}} (\mathbf{I} - \gamma P_\pi)^\mathsf{T} u_a = \alpha.$$

2.3 Value Functions and Policies

We start the section by defining the basic properties of value functions and then discuss how a value function can be used to construct a policy. These properties are important in defining the objectives in calculating a value function. The approximate value function $\tilde{v}$ in this section is an arbitrary estimate of the optimal value function v^*.

Value functions serve to simplify the solution of MDPs because they are easier to calculate and analyze than policies. Finding the optimal value function for an MDP corresponds to finding a fixed point of the *nonlinear Bellman operator* (Bellman, 1957).

Definition 2.6 ((Puterman, 2005))**.** The Bellman operator $L : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ and the value function update $L_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ for a policy π are defined as:

$$\begin{aligned} L_\pi v &= \gamma P_\pi v + r_\pi \\ L v &= \max_{\pi \in \Pi} L_\pi v. \end{aligned}$$

The operator L is well-defined because the maximization can be decomposed state-wise. That is $Lv \geq L_\pi v$ for all policies $\pi \in \Pi$. Notice that L is a *non-linear operator* and L_π is an *affine operator* (a linear operator offset by a constant).

A value function is optimal when it is stationary with respect to the Bellman operator.

Theorem 2.7 ((Bellman, 1957))**.** *A value function v^* is optimal if and only if $v^* = Lv^*$. Moreover, v^* is unique and satisfies $v^* \geq v_\pi$.*

The proof of the theorem can be found in Section C.2. The proof of this basic property is in Section C.2. It illustrates well the concepts used in other proofs in the thesis.

Because the ultimate goal of solving an MDP is to compute a good policy, it is necessary to be able to compute a policy from a value function. The simplest method is to take the greedy policy with respect to the value function. A greedy policy takes in each state the action that maximizes the expected value when transiting to the following state.

Definition 2.8 (Greedy Policy)**.** A policy π is *greedy* with respect to a value function v when

$$\pi(s) = \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') v(s') = \arg\max_{a \in \mathcal{A}} \mathbf{1}_s^\mathsf{T} \left(r_a + \gamma P_a v \right),$$

and is greedy with respect to an action-value function q when

$$\pi(s) = \arg\max_{a \in \mathcal{A}} q(s,a).$$

The following propositions summarize the main properties of greedy policies.

Proposition 2.9. *The policy π greedy for a value-function v satisfies $Lv = L_\pi v \geq L_{\pi'} v$ for all policies $\pi' \in \Pi$. In addition, the greedy policy with respect to the optimal value function v^* is an optimal policy.*

The proof of the proposition can be found in Section C.3.

Most practical MDPs are too large for the optimal value function to be computed precisely. In these cases, we calculate an approximate value function $\tilde{v}$ and take the greedy policy π with respect to it. The quality of such a policy can be evaluated from its value function v_π in one of the following two main ways.

Definition 2.10 (Policy Loss)**.** Let π be a policy computed from value function approximation. The *average policy loss* measures the expected loss of π and is defined as:

$$\|v^* - v_\pi\|_{1,\alpha} = \alpha^{\mathsf{T}} v^* - \alpha^{\mathsf{T}} v_\pi \tag{2.1}$$

The *robust policy loss* measures the robust policy loss of π and is defined as:

$$\|v^* - v_\pi\|_\infty = \max_{s\in\mathcal{S}} |v^*(s) - v_\pi(s)| \tag{2.2}$$

The average policy loss captures the total loss of discounted average reward when following the policy π instead of the optimal policy, assuming the initial distribution. The robust policy loss ignores the initial distribution and measures the difference for the worst-case initial distribution.

Taking the greedy policy with respect to a value function is the simplest method for choosing a policy. There are other — more computationally intensive — methods that can often lead to much better performance, but are harder to construct and analyze. We discuss these and other methods in more detail in Chapter 11.

The methods for constructing policies from value functions can be divided into two main classes based on the effect of the value function error, as Chapter 11 describes in more detail. In the first class of methods, the computational complexity increases with a value function error, but solution quality is unaffected. A* and other classic search methods are included in this class (Russell & Norvig, 2003). In the second class of methods, the solution quality decreases with value function error, but the computational complexity is unaffected. Greedy policies are an example of such a method. In the remainder of the thesis, we focus on greedy policies because they are easy to study, can be easily constructed, and often work well.

A crucial concept in evaluating the quality of a value function with respect to the greedy policy is the Bellman residual, which is defined as follows.

Definition 2.11 (Bellman residual)**.** The *Bellman residual* of a value function v is a vector defined as $v - Lv$.

The Bellman residual can be easily estimated from data, and is used in bounds on the policy loss of greedy policies. Most methods that approximate the value function are at least loosely based on minimization of a function of the Bellman residual.

In many of the methods that we study, it is advantageous to restrict the value functions so that their Bellman residual must be non-negative, or at least bounded from below. We call such value functions transitive-feasible and define them as follows.

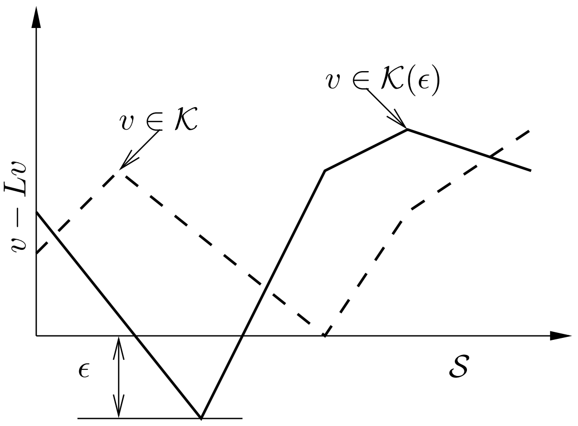

Figure 2.1. Transitive-feasible value functions in an MDP with a linear state-space.

Definition 2.12. A value function is *transitive-feasible* when $v \geq Lv$. The set of transitive-feasible value functions is:

$$\mathcal{K} = \{v \in \mathbb{R}^{|\mathcal{S}|} \,\big|\, v \geq Lv\}.$$

Assume an arbitrary $\epsilon \geq 0$. The set of ϵ-*transitive-feasible* value functions is defined as follows:

$$\mathcal{K}(\epsilon) = \{v \in \mathbb{R}^{|\mathcal{S}|} \,\big|\, v \geq Lv - \epsilon \mathbf{1}\}.$$

Notice that the optimal value function v^* is transitive-feasible, which follows directly from Theorem 2.7. Transitive-feasible value functions are illustrated in Figure 2.1. The following lemma summarizes the main importance of transitive-feasible value functions:

Lemma 2.13. *Transitive feasible value functions are an upper bound on the optimal value function. Assume an ϵ-transitive-feasible value function $v \in \mathcal{K}(\epsilon)$. Then:*

$$v \geq v^* - \frac{\epsilon}{1-\gamma}\mathbf{1}.$$

The proof of the lemma can be found in Section C.2.

Another important property of transitive-feasible value functions follows.

Proposition 2.14. *The set $\mathcal{K}$ of transitive-feasible value functions is convex. That is for any $v_1, v_2 \in \mathcal{K}$ and any $\beta \in [0,1]$ also $\beta v_1 + (1-\beta)v_2 \in \mathcal{K}$.*

The proof of the proposition can be found in Section C.2.

The crucial property of approximate value functions is the quality of the corresponding greedy policy. The robust policy loss can be bounded as follows.

Theorem 2.15. *Let $\tilde{v}$ be the approximate value function, and v_π be a value function of an arbitrary policy π. Then:*

$$\|v^* - v_\pi\|_\infty \leq \frac{1}{1-\gamma}\|\tilde{v} - L_\pi \tilde{v}\|_\infty + \|\tilde{v} - v^*\|_\infty$$

$$\|v^* - v_\pi\|_\infty \leq \frac{2}{1-\gamma}\|\tilde{v} - L_\pi \tilde{v}\|_\infty$$

The proof of the theorem can be found in Section C.3. This theorem extends the classical bounds on policy loss (Williams & Baird, 1994). The following theorem states the bounds for the greedy policy in particular.

Theorem 2.16 (Robust Policy Loss). *Let π be the policy greedy with respect to $\tilde{v}$. Then:*

$$\|v^* - v_\pi\|_\infty \le \frac{2}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty .$$

In addition, if $\tilde{v} \in \mathcal{K}$, the policy loss is minimized for the greedy policy and:

$$\|v^* - v_\pi\|_\infty \le \frac{1}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty .$$

The proof of the theorem can be found in Section C.3.

The bounds above ignore the initial distribution. When the initial distribution is known, bounds on the expected policy loss can be used.

Theorem 2.17 (Expected Policy Loss). *Let π be a greedy policy with respect to a value function $\tilde{v}$ and let the state-action visitation frequencies of π be bounded as $\underline{u} \le u_\pi \le \bar{u}$. Then:*

$$\begin{aligned}\|v^* - v_\pi\|_{1,\alpha} &= \alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} + u_\pi^\mathsf{T}(\tilde{v} - L\tilde{v}) \\ &\le \alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} + \underline{u}^\mathsf{T}\left[\tilde{v} - L\tilde{v}\right]_- + \bar{u}^\mathsf{T}\left[\tilde{v} - L\tilde{v}\right]_+ .\end{aligned}$$

The state-visitation frequency u_π depends on the initial distribution α, unlike v^*. In addition, when $\tilde{v} \in \mathcal{K}$, the bound is:

$$\begin{aligned}\|v^* - v_\pi\|_{1,\alpha} &\le -\|v^* - \tilde{v}\|_{1,\alpha} + \|\tilde{v} - L\tilde{v}\|_{1,\bar{u}} \\ \|v^* - v_\pi\|_{1,\alpha} &\le -\|v^* - \tilde{v}\|_{1,\alpha} + \frac{1}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty\end{aligned}$$

The proof of the theorem can be found in Section C.3. The proof is based on the complementary slackness principle in linear programs (Mendelssohn, 1980; Shetty & Taylor, 1987; Zipkin, 1977). Notice that the bounds in Theorem 2.17 can be minimized even without knowing v^*. The optimal value function v^* is independent of the approximate value function $\tilde{v}$ and the greedy policy π depends only on $\tilde{v}$.

Remark 2.18. The bounds in Theorem 2.17 generalize the bounds of Theorem 1.3 in (de Farias, 2002). Those bounds state that whenever $v \in \mathcal{K}$:

$$\|v^* - v_\pi\|_{1,\alpha} \le \frac{1}{1-\gamma}\|v^* - \tilde{v}\|_{1,(1-\gamma)u} .$$

This bound is a special case of Theorem 2.17 because:

$$\|\tilde{v} - L\tilde{v}\|_{1,u} \le \|v^* - \tilde{v}\|_{1,u} \le \frac{1}{1-\gamma}\|v^* - \tilde{v}\|_{1,(1-\gamma)u},$$

from $v^* \le L\tilde{v} \le \tilde{v}$ and $\alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} \le 0$. The proof of Theorem 2.17 also simplifies the proof of Theorem 1.3 in (de Farias, 2002)

The bounds from the remark above can be further tightened and revised as the following theorem shows. We use this new bound later to improve the standard ALP formulation.

Theorem 2.19 (Expected Policy Loss). *Let π be a greedy policy with respect to a value function $\tilde{v}$ and let the state-action visitation frequencies of π be bounded as $\underline{u} \leq u_\pi \leq \bar{u}$. Then:*

$$\|v^* - v_\pi\|_{1,\alpha} \leq \left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)(\tilde{v} - v^*) + \bar{u}^{\mathsf{T}}\left[L\tilde{v} - \tilde{v}\right]_+,$$

where $P^* = P_{\pi^*}$. The state-visitation frequency u_π depends on the initial distribution α, unlike v^*. In addition, when $\tilde{v} \in \mathcal{K}$, the bound can be simplified to:

$$\|v^* - v_\pi\|_{1,\alpha} \leq \bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*)(\tilde{v} - v^*)$$

This simplification is, however, looser.

The proof of the theorem can be found in Section C.3.

Notice the significant difference between Theorems 2.19 and 2.17. Theorem 2.19 involves the term $[L\tilde{v} - \tilde{v}]_+$, while in Theorem 2.17 it is reversed to be $\tilde{v} - [L\tilde{v}]_+$.

The bounds above play in important role in the approaches that we propose. Chapter 4 shows that approximate linear programming minimizes bounds on the expected policy loss in Theorems 2.17 and 2.19. However, it only minimizes loose upper bounds. Then, Chapter 5 shows that the tighter bounds on Theorems 2.17 and 2.16 can be optimized using approximate bilinear programming.

2.4 Approximately Solving Markov Decision Processes

Most interesting MDP problems are too large to be solved precisely and must be approximated. The methods for approximately solving Markov decision processes can be divided into two main types: 1) *policy search* methods, and 2) *approximate dynamic programming* methods. This thesis focuses on approximate dynamic programming.

Policy search methods rely on local search in a restricted policy space. The policy may be represented, for example, as a finite-state controller (Stanley & Miikkulainen, 2004) or as a greedy policy with respect to an approximate value function (Szita & Lorincz, 2006). Policy search methods have achieved impressive results in such domains as Tetris (Szita & Lorincz, 2006) and helicopter control (Abbeel et al., 2006). However, they are notoriously hard to analyze. We are not aware of any theoretical guarantees regarding the quality of the solution.

Approximate dynamic programming (ADP) methods, also known as value function approximation, first calculate the value function approximately (Bertsekas & Ioffe, 1997; Powell, 2007a; Sutton & Barto, 1998). The policy is then calculated from this value function. The advantage of value function approximation is the it is easy to determine the quality of a value function using samples, while determining a quality of a policy usually requires extensive simulation. We discuss these properties in greater detail in Section 2.3.

A basic setup of value function approximation is depicted in Figure 2.2. The ovals represent inputs, the rectangles represent computational components, and the arrows represent information flow. The input "Features" represents functions that assign a set of real values to each state, as described in Section 2.4.1. The input "Samples" represents a set of

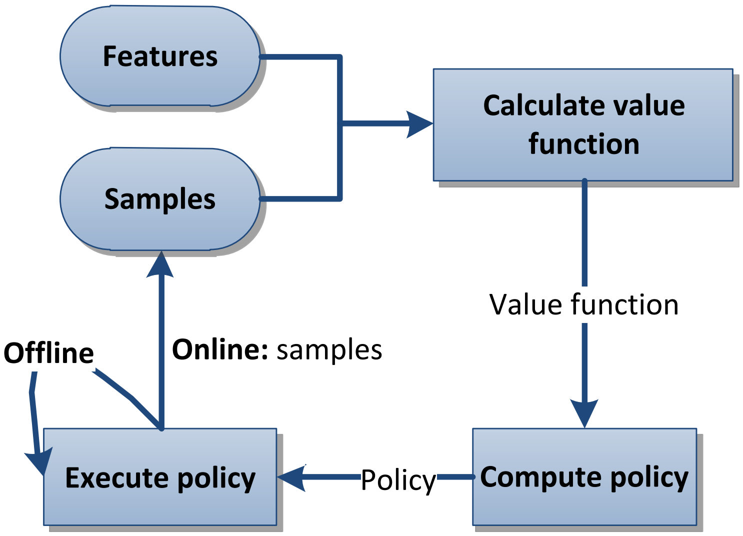

Figure 2.2. Approximate Dynamic Programming Framework.

simple sequences of states and actions generated using the transition model of an MDP, as described in Section 2.4.2.

A significant part of the thesis is devoted to studying methods that calculate the value function from the samples and state features. These methods are described and analyzed in Chapters 3, 4, and 5. A crucial consideration in the development of the methods is the way in which a policy is constructed from a value function.

Value function approximation methods can be classified based on the source of samples into *online* and *offline* methods as Figure 2.2 shows. Online methods interleave execution of a calculated policy with sample gathering and value function approximation. As a result, a new policy may be often calculated during execution. Offline methods use a fixed number of samples gathered earlier, prior to plan execution. They are simpler to analyze and implement than online methods, but may perform worse due to fewer available samples. To simplify the analysis, our focus is on the offline methods, and we indicate the potential difficulties with extending the methods to the online ones when appropriate.

2.4.1 Features

The set of the state features is a necessary component of value function approximation. These features must be supplied in advance and must roughly capture the properties of the problem. For each state s, we define a vector $\phi(s)$ of features and denote $\phi_i : \mathcal{S} \to \mathbb{R}$ to be a function that maps states to the value feature i:

$$\phi_i(s) = (\phi(s))_i.$$

The desirable properties of features to be provided depend strongly on the algorithm, samples, and attributes of the problem, and their best choice is not yet fully understood. The feature function ϕ_i can also be treated as a vector, similarly to the value function v. We use $|\phi|$ to denote the number of features.

Value function approximation methods combines the features into a value function. The main purpose is to limit the possible value functions that can be represented, as the following shows.

Definition 2.20. Assume a given *convex* polyhedral set $\mathcal{M} \subseteq \mathbb{R}^{|\mathcal{S}|}$. A value function v is *representable* (in $\mathcal{M}$) if $v \in \mathcal{M}$.

This definition captures the basic properties and we in general assume its specific instantiations. In particular, we generally assume that $\mathcal{M}$ can be represented using a set of linear constraints, although most approaches we propose and study can be easily generalized to quadratic functions.

Many complex methods that combine features into a value function have been developed, such as neural networks and genetic algorithms (Bertsekas & Ioffe, 1997). Most of these complex methods are extremely hard to analyze, computationally complex, and hard to use. A simpler, and more common, method is *linear value function approximation.* In linear value function approximation, the value function is represented as a linear combination of *nonlinear features* $\phi(s)$. Linear value function approximation is easy to apply and analyze, and therefore commonly used.

It is helpful to represent linear value function approximation in terms of matrices. To do that, let the matrix $\Phi : |\mathcal{S}| \times m$ represent the features, where m is the number of features. The feature matrix Φ, also known as *basis*, has the features of the states $\phi(s)$ as rows:

$$\Phi = \begin{pmatrix} - & \phi(s_1)^{\mathsf{T}} & - \\ - & \phi(s_2)^{\mathsf{T}} & - \\ & \vdots & \end{pmatrix} \qquad\qquad \Phi = \begin{pmatrix} | & | & \\ \phi_1 & \phi_2 & \dots \\ | & | & \end{pmatrix}$$

The value function v is then represented as $v = \Phi x$ and $\mathcal{M} = \operatorname{colspan}(\Phi)$

Generally, it is desirable that the number of all features is relatively small because of two main reasons. First, a limited set of features enables generalization from an incomplete set of samples. Second, it reduces computational complexity since it restricts the space of representable value functions. When the number of features is large, it is possible to achieve these goals using *regularization.* Regularization restricts the coefficients x in $v = \Phi x$ using a norm as: $\|x\| \le \psi$. Therefore, we consider the following two types of representation:

Linear space: $\mathcal{M} = \{v \in \mathbb{R}^{|\mathcal{S}|} \mid v = \Phi x\}$

Regularized: $\mathcal{M}(\psi) = \{v \in \mathbb{R}^{|\mathcal{S}|} \mid v = \Phi x,\ \Omega(x) \le \psi\}$, where $\Omega : \mathbb{R}^m \to \mathbb{R}$ is a *convex* regularization function and m is the number of features.

When not necessary, we omit ψ in the notation of $\mathcal{M}(\psi)$. Methods that we propose require the following standard assumption (Schweitzer & Seidmann, 1985).

Assumption 2.21. All multiples of the constant vector $\mathbf{1}$ are representable in $\mathcal{M}$. That is, for all $k \in \mathbb{R}$ we have that $k\mathbf{1} \in \mathcal{M}$.

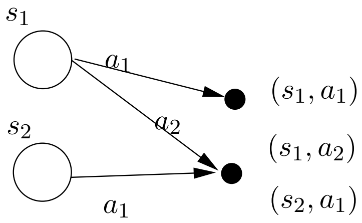

Figure 2.3. Example of equivalence of pairs of state-action values.

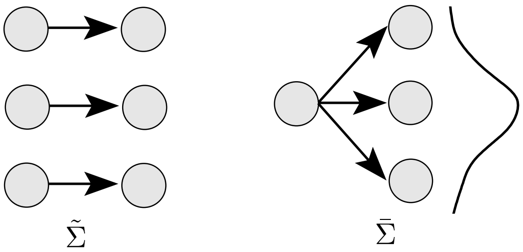

Figure 2.4. Sample types

We implicitly assume that the first column of Φ — that is ϕ_1 — is the constant vector $\mathbf{1}$. Assumption 2.21 is satisfied when the first column of Φ is $\mathbf{1}$ — or a constant feature — and the regularization (if any) does not place any penalty on this feature. The influence of the constant features is typically negligible because adding a constant to the value function does not influence the policy as Lemma C.5 shows.

The state-action value function q can be approximated similarly. The main difference is that the function q is approximated for all actions $a \in \mathcal{A}$. That is for all actions $a \in \mathcal{A}$:

$$q(a) = \Phi_a x_a.$$

Notice that Φ_a and x_a may be the same for multiple states or actions.

Policies, like value functions, can be represented as vectors. That is, a policy π can be represented as a vectors over state-action pairs.

2.4.2 Samples

In most practical problems, the number of states is too large to be explicitly enumerated. Therefore, even though the value function is restricted as described in Section 2.4.1, the problem cannot be solved optimally. The approach taken in reinforcement learning is to sample a limited number of states, actions, and their transitions to approximately calculate the value function. It is possible to rely on state samples because the value function is restricted to the representable set $\mathcal{M}$. Issues raised by sampling are addressed in greater detail in Chapter 9.

Samples are usually used to approximate the Bellman residual. First, we show a formal definition of the samples and then show how to use them.

Definition 2.22. *One-step simple samples* are defined as:

$$\tilde{\Sigma} \subseteq \{(s, a, (s_1 \ldots s_n), r(s, a)) \,|\, s \in \mathcal{S},\ a \in \mathcal{A}\},$$

where $s_1 \ldots s_n$ are selected i.i.d. from the distribution $P(s, a)$ for *every* s, a *independently.*

Definition 2.23. *One-step samples with expectation* are defined as follows:

$$\bar{\Sigma} \subseteq \{(s, a, P(s, a), r(s, a)) \,|\, s \in \mathcal{S},\ a \in \mathcal{A}\}.$$

Notice that $\bar{\Sigma}$ are more informative than $\tilde{\Sigma}$, but are often unavailable. Membership of states in the samples is denoted simply as $s \in \Sigma$ or $(s, a) \in \Sigma$ with the remaining variables, such as $r(s, a)$ considered to be available implicitly. Examples of these samples are sketched in Figure 2.4.

We use $|\bar{\Sigma}|_s$ to denote the number of samples in terms of distinct states, and $|\bar{\Sigma}|_a$ to denote the number of samples in terms of state–action pairs. The same notation is used for $\tilde{\Sigma}$.

As defined here, the samples do not repeat for states and actions, which differs from the traditional sampling assumptions in machine learning. Usually, the samples are assumed to be drawn with repetition from a given distribution. In comparison, we do not assume a distribution over states and actions.

The sampling models vary significantly in various domains. In some domains, it may be very easy and cheap to gather samples. In the blood inventory management problem, the model of the problem has been constructed based on historical statistics. The focus of this work is on problems with a model available. This fact simplifies many of the assumptions on the source and structure of the samples, since they can be essentially generated for an arbitrary number of states. In general reinforcement learning, often the only option is to gather samples during the execution. Much work has focused on defining setting and sample types appropriate in these settings (Kakade, 2003), and we discuss some of them in Chapter 9.

In *online* reinforcement learning algorithms (see Figure 2.2), the samples are generated during the execution. It is important then to determine the tradeoff between exploration and exploitation. We, however, consider *offline* algorithms, in which the samples are generated in advance. While it is still desirable to minimize the number of samples needed there is no tradeoff with the final performance. Offline methods are much easier to analyze and are more appropriate in many planning settings.

The samples, as mentioned above, are used to approximate the Bellman operator and the set of transitive-feasible value functions.

Definition 2.24. The *sampled Bellman operator* and the corresponding set of sampled transitive-feasible functions are defined as:

$$(\bar{L}(v))(\bar{s}) = \begin{cases} \max_{\{a \,|\, (\bar{s}, a) \in \bar{\Sigma}\}} r(\bar{s}, a) + \gamma \sum_{s' \in \mathcal{S}} P(\bar{s}, a, s') v(s') & \text{when } \bar{s} \in \bar{\Sigma} \\ -\infty & \text{otherwise} \end{cases} \tag{2.3}$$

$$\bar{\mathcal{K}} = \left\{ v \,\middle|\, \forall s \in \mathcal{S} \ \ v(s) \geq (\bar{L}v)(s) \right\} \tag{2.4}$$

The less-informative $\tilde{\Sigma}$ can be used as follows.

Definition 2.25. The *estimated Bellman operator* and the corresponding set of estimated transitive-feasible functions are defined as:

$$(\tilde{L}(v))(\bar{s}) = \begin{cases} \max_{\{a \,|\, (\bar{s},a)\in\tilde{\Sigma}\}} r(\bar{s},a) + \gamma\frac{1}{n}\sum_{i=1}^{n} v(s_i) & \text{when } \forall \bar{s} \in \tilde{\Sigma} \\ -\infty & \text{otherwise} \end{cases} \tag{2.5}$$

$$\tilde{\mathcal{K}} = \left\{ v \,\middle|\, \forall s \in \mathcal{S} \;\; v(s) \geq (\tilde{L}v)(s) \right\} \tag{2.6}$$

Notice that operators $\tilde{L}$ and $\bar{L}$ map value functions to a subset of all states — only states that are sampled. The values for other states are assumed to be *undefined.*

The samples can also be used to create an approximation of the initial distribution, or the distribution of visitation-frequencies of a given policy. The estimated initial distribution $\bar{\alpha}$ is defined as:

$$\bar{\alpha}(s) = \begin{cases} \alpha(s) & s \in \bar{\Sigma} \\ 0 & \text{otherwise} \end{cases}$$

Although we define above the sampled operators and distributions directly, in applications only their estimated versions are typically calculated. That means calculating $\bar{\alpha}^{\mathsf{T}}\Phi$ instead of estimating $\bar{\alpha}$ first. The generic definitions above help to generalize the analysis to various types of approximate representations.

To define bounds on the sampling behavior, we propose the following assumptions. These assumptions are intentionally generic to apply to a wide range of scenarios. Chapter 9 examines some more-specific sampling conditions and their implications in practice. Note in particular that the assumptions apply only to value functions that are representable. The first assumption limits the error due to missing transitions in the sampled Bellman operator $\bar{L}$.

Assumption 2.26 (State Selection Behavior)**.** The representable value functions satisfy for some ϵ_p:

$$\mathcal{K} \cap \mathcal{M} \subseteq \bar{\mathcal{K}} \cap \mathcal{M} \subseteq \mathcal{K}(\epsilon_p) \cap \mathcal{M}.$$

When $\mathcal{M}(\psi)$ is a function of ψ then we write $\epsilon_p(\psi)$ to denote the dependence.

The constant ϵ_p bounds the potential violation of the Bellman residual on states that are not provided as a part of the sample. In addition, all value functions that are transitive-feasible for the full Bellman operator are transitive-feasible in the sampled version; the sampling only removes constraints on the set.

The second assumptions bounds the error due to sampling non-uniformity.

Assumption 2.27 (Uniform Sampling Behavior)**.** For all representable value functions $v \in \mathcal{M}$:

$$|(\alpha - \bar{\alpha})^{\mathsf{T}} v| \leq \epsilon_c.$$

When $\mathcal{M}(\psi)$ is a function of ψ then we write $\epsilon_c(\psi)$ to denote the dependence.

This assumption is necessary to estimate the initial distribution from samples. This is important only in some of the methods that we study and strongly depends on the actual domain. For example, the initial distribution is very easy to estimate when there is only single initial state. The constant ϵ_c essentially represents the maximal difference between the true expected return and the sampled expected return for a representable value function.

The third assumption quantifies the error on the estimated Bellman operator $\tilde{L}$.

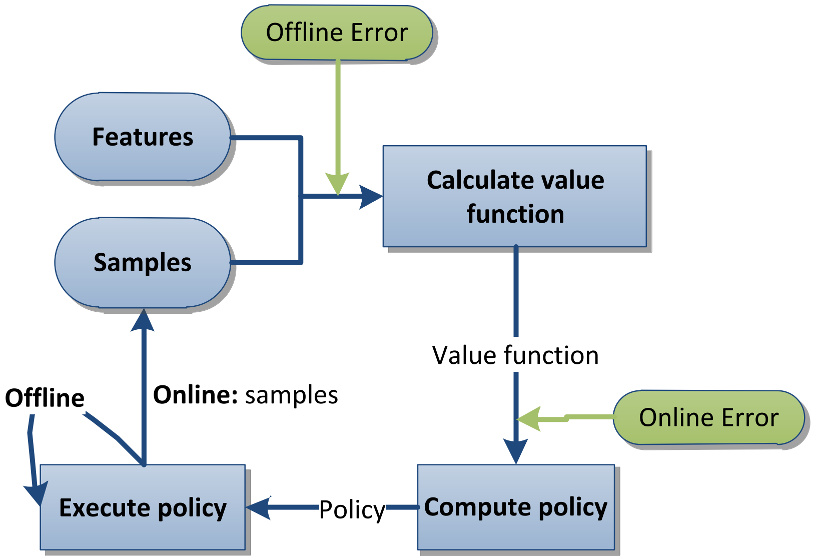

Figure 2.5. Online and offline errors in value function approximation.

Assumption 2.28 (Transition Estimation Behavior)**.** The representable value functions satisfy for some ϵ_s:

$$\bar{\mathcal{K}}(-\epsilon_s) \cap \mathcal{M} \subseteq \tilde{\mathcal{K}} \cap \mathcal{M} \subseteq \bar{\mathcal{K}}(\epsilon_s) \cap \mathcal{M},$$

where $\bar{\Sigma}$ and $\tilde{\Sigma}$ (and therefore $\bar{\mathcal{K}}$ and $\tilde{\mathcal{K}}$) are defined for identical sets of states. When $\mathcal{M}(\psi)$ is a function of ψ then we write $\epsilon_s(\psi)$ to denote the dependence.

The constant ϵ limits the maximal error in estimating the constraints from samples. This error is zero when the samples $\bar{\Sigma}$ are available. Note that unlike ϵ_p, the error ϵ_s applies both to the left and right sides of the subset definition.

2.5 Approximation Error: Online and Offline

One of the most important issues with approximating value function is estimating the approximation error. The approximation error is the difference between the return of the optimal policy π^* and the approximate policy $\tilde{\pi}$, as bounded by (2.2) or (2.1). Approximation errors are important for two main reasons: 1) they guide development and analysis of algorithms, and 2) they provide guarantees for their quality.

We consider two main types of approximation error: *online* and *offline*. These errors are evaluated at different stages of the approximation as Figure 2.5 shows. The online error measures the quality of the solution — the value function — and serves to guide and analyze the approximate algorithms. The offline error measures how well the algorithm minimizes the online error bound and serves to provide guarantees on the solution quality. Notice that the offline bounds rely on the representation and samples only and does not use the approximate value function. Other types of errors are clearly also possible, such as an error of the policy, or an error that provides guarantees independent of samples. In case

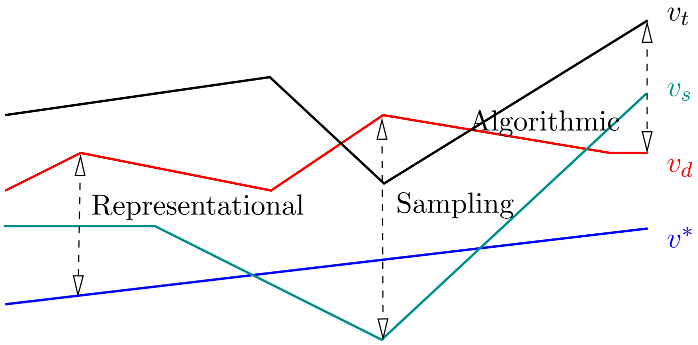

Figure 2.6. Value functions and the components of the approximation error.

that a policy is constructed from the value function using heuristic search, as Chapter 11 discusses, it makes sense to consider the online bound as a function of the policy, not the value function.

The algorithms in this work minimize the online bounds on value function approximation, as discussed in Section 2.3. The remainder of the thesis therefore focuses on developing such algorithms and an analysis of the offline error bound. Therefore, when referring to a approximation error, we generally mean the offline error bound.

The offline approximation error is due to the inherent limitations on the availability of computational time, sufficient number of samples, and a precise model. If unlimited resources were available, it would be possible to solve any MDP problem accurately. To provide a deeper analysis of these tradeoffs, we decompose the approximation error into three parts depending on the basic reasons that cause it. This decomposition is more conceptual than formal in order to apply to all algorithms. It is, however, easy to write down formally for the particular algorithms.

The offline approximation error can be decomposed into three main parts as illustrated in Figure 2.6. The figure shows approximate value functions on the state space represented by the x-axis. The optimal value function is denoted as v^*. The value function that minimizes the online error bound — on one of (2.1) or (2.2) — for all samples is denoted as v_d and its error is the **representational error**. Notice that a representational error is different from the online approximation error.

It is typically impossible to compute v_d, since it relies on the precise version of the Bellman residual. The solution is based instead on approximate Bellman operators, calculated from the samples. The solution that minimizes the online error bound and only relies on the estimated Bellman operator is denoted as v_s. The difference between v_s and v_d is the **sampling error**.

The sampling error can be further divided into two main components, based on its source. The first component is the *state selection error*. State selection error arises from the incomplete sample of states in sampled Bellman operator $\bar{L}$. The error is caused by the missing state transitions. As we show in Chapters 4 and 5, the state selection error can be bounded based on Assumptions 2.26 and 2.27.

The second component of sampling error is the *transition estimation error*. Transition estimation error is caused by the imperfect estimation of the estimated Bellman operator

$\tilde{L}$. In simple domains with known transition models (when sample $\bar{\Sigma}$ are available), this is not typically an issue. In complex domains it may be impossible to express the constraints analytically even when the transition model is known. As we show in Chapters 4 and 5, the transition estimation error can be bounded using Assumption 2.28.

Many value function approximation algorithms do not calculate the closest approximation using the estimated Bellman operator. One of the reasons, as we show in Chapter 5, is that calculating such an approximation is NP complete. The value function calculated by the algorithm is denoted as v_t, and the difference between v_s and v_t is the **algorithmic error**.

2.6 Contributions

Most of the concepts discussed in this chapter have been introduced in previous work. The main contributions is a slight extension of the bounds on robust policy loss in Theorems 2.15 and 2.16. The bound on the expected policy loss in Theorem 2.17 is new. It extends and simplifies bounds previously proposed in (de Farias, 2002) (Theorem 1.3). The decomposition of the approximation error into the three component is also new.

CHAPTER 3

ITERATIVE VALUE FUNCTION APPROXIMATION

This chapter overviews the standard iterative methods for value function approximation. These methods include some of the most popular algorithms in reinforcement learning, such as least-squares policy iteration (Lagoudakis & Parr, 2003), λ-policy iteration (Bertsekas & Ioffe, 1997), fitted value iteration (Szepesvari & Munos, 2005; Munos & Szepesvari, 2008), fitted policy iteration (Antos, Szepesvri, & Munos, 2008), Q-learning, and SARSA (Sutton & Barto, 1998).

We also describe basic offline error upper bounds on iterative algorithms and show that they may be very loose. Lower bounds on their performance demonstrate that the basic iterative algorithms cannot offer guarantees as strong as the optimization–based algorithms, which are described later. Approaches that rely on these bounds to improve performance have been studied in (Petrik & Scherrer, 2008), but we do not discuss them in detail here since iterative methods are not the main focus of this work.

3.1 Basic Algorithms

Basic iterative techniques for solving MDPs precisely include *value iteration*, and *policy iteration* and are based on calculating the value function. We only describe the basic versions of value and policy iterations; in practice, the methods are modified to improve their performance (Puterman, 2005). While these methods are iterative, their convergence to the optimal solution is guaranteed because L is a contraction (Theorem C.9) in the L_∞ norm. Linear programming can also be used to solve MDPs. Because the formulation is closely related to approximate linear programming, we describe it in Chapter 4.

Iterative value function approximation algorithms are variations of the exact MDP algorithms. Hence, they can be categorized as approximate value iteration and approximate policy iteration. We discuss these in greater detail below. The ideas of approximate value iteration could be traced to Bellman (Bellman, 1957), which was followed up by many additional research efforts (Bertsekas & Tsitsiklis, 1996; Powell, 2007a; Sutton & Barto, 1998).

The iterative algorithms for value function approximation are almost always based on domain samples. To keep the analysis simple, we assume that the set of samples for *all* states is available, as described in Section 2.4.2. That means that the exact Bellman operator L is available and can be used. One of the difficulties with iterative algorithms is that their sampling behavior is hard to study.

A simple variant of *approximate value iteration* (Powell, 2007a; Munos, 2007) is depicted in algorithm 3.1. Approximate value iteration is also known as fitted value iteration (Munos

Algorithm 3.1: Approximate value iteration: Value function approximation

```
$k \leftarrow 0$ ;
$v_k \leftarrow \mathbf{0}$ ;
while $\|v_k - v_{k-1}\|_\infty > \epsilon$ do
    $v_{k+1} \leftarrow \Phi \left(\Phi^\mathsf{T}\Phi\right)^{-1} \Phi^\mathsf{T} L v_k$ ;
    $k \leftarrow k + 1$ ;
return $v_k$;
```

& Szepesvari, 2008), and is closely related to temporal difference learning and fitted value iteration (Maei, Szepesvari, Bhatnagar, Precup, Silver, & Sutton, 2009). While the algorithm shown assumes that all states and actions are known, in practice these values could be based on samples.

Algorithm 3.2: Approximate value iteration: State-action value function approximation

```
$k \leftarrow 0$ ;
$q_k \leftarrow \mathbf{0}$ ;
while $\|q_k(\pi_k) - q_{k-1}(\pi_{k-1})\|_\infty > \epsilon$ do
    $\pi_k(s) \leftarrow \arg\max_{a \in \mathcal{A}} q_k(s,a) \quad \forall s \in \mathcal{S}$ ;        // Greedy policy
    $q_{k+1}(\pi_k) \leftarrow \Phi_{\pi_k} \left(\Phi_{\pi_k}^\mathsf{T}\Phi_{\pi_k}\right)^{-1} \Phi_{\pi_k}^\mathsf{T} L q_k(\pi_k)$ ;
    $k \leftarrow k + 1$ ;
return $q_k$;
```

Approximate value iteration described above requires that all (or most) available actions are sampled for every state or that they can be easily generated. This is only feasible when a model of the MDP is available. It is often desirable to be able to solve the MDP with a single action sampled for every state. The typical approach to estimate the state-action value function q instead. algorithm 3.2 shows an approximate value iteration that approximates the state-action value function q. Here Φ_{π_k} corresponds to the features for the policy π_k.

Algorithm 3.3: Approximate policy iteration: Value function approximation

```
$k \leftarrow 0$;
$v_k \leftarrow \mathbf{0}$ ;
while $\pi_k \neq \pi_{k-1}$ do
    $\pi_k(s) \leftarrow \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma P(s,a)^\mathsf{T} v_k \quad \forall s \in \mathcal{S}$ ;        // Greedy policy
    $v_k \leftarrow \mathcal{Z}(\pi_k)$ ;
    $k \leftarrow k + 1$;
return $v_k$;
```

A simple variant of *approximate policy iteration* (API) is summarized in algorithm 3.3 (Powell, 2007a; Munos, 2003). The function $\mathcal{Z}(\pi)$ denotes the method used to approximate the value function for a policy π. The two most commonly used value function approximation methods are the *Bellman residual approximation* (3.1) and *least-squares approximation* (3.2):

$$\begin{aligned} \mathcal{Z}(\pi) &= \arg\min_{v\in\mathcal{M}} \|(\mathbf{I}-\gamma P_\pi)v - r_\pi\|_2 && (3.1)\\ &= \Phi\left(\Phi^\mathsf{T}(\mathbf{I}-\gamma P_\pi)^\mathsf{T}(\mathbf{I}-\gamma P_\pi)\Phi\right)^{-1}\Phi^\mathsf{T}(\mathbf{I}-\gamma P_\pi)^\mathsf{T} r_\pi \\ \mathcal{Z}(\pi) &= \left(\mathbf{I}-\gamma\Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T}P_\pi\right)^{-1}\Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T} r_\pi && (3.2)\\ &= \Phi\left(\Phi^\mathsf{T}(\mathbf{I}-\gamma P)\Phi\right)^{-1}\Phi^\mathsf{T} r. && (3.3) \end{aligned}$$

The definitions above assume that the columns of Φ are linearly independent. Note that the matrix in (3.2) may not be invertible (Scherrer, 2010).

The following proposition relates the least-squares formulation with the Bellman residual. We use this property to motivate and compare approximate bilinear formulations. For a more detailed analysis, see for example Johns, Petrik, and Mahadevan (2009).

Proposition 3.1. *The least-squares formulation of $\mathcal{Z}$ in (3.2) minimizes the L_2 norm of a projection of the Bellman residual. Let v be the least-squares solution, then it satisfies:*

$$v = \arg\min_{v\in\mathcal{M}} \|\Phi^\mathsf{T}(L_\pi v - v)\|_2.$$

The proof of the proposition can be found in Section C.4.

The approximations above are based on the L_2 norm and are common in practice, because they are easy to compute and often lead to good results. From the perspective of theoretical analysis, it is more convenient to approximate the value function using the L_∞ norm as follows:

$$\mathcal{Z}(\pi) \in \arg\min_{v\in\mathcal{M}} \|(\mathbf{I}-\gamma P_\pi)v - r_\pi\|_\infty \tag{3.4}$$

This approximation can be calculated easily using the following linear program.

$$\begin{aligned} \min_{\phi,v} \quad & \phi \\ \text{s.t.} \quad & (\mathbf{I}-\gamma P_\pi)v + \mathbf{1}\phi \geq r_\pi \\ & -(\mathbf{I}-\gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{aligned} \tag{3.5}$$

We refer to approximate policy iteration with this approximation as L_∞-API.

Approximate policy iteration can also be expressed in terms of state-action value function, as algorithm 3.4 shows. This algorithm represents a simplified version of *least-squares policy iteration* (Lagoudakis & Parr, 2003).

In the above description of approximate dynamic programming, we assumed that the value function is approximated for all states and actions. This is impossible in practice due to the size of the MDP. Instead, approximate dynamic programming relies only on a subset of states and actions which are provided as samples, as described in Section 2.4.2.

Approximate dynamic programming is not guaranteed to converge in general and its analysis is typically in terms of limit behavior. We discuss this behavior and corresponding bounds in the following section.

Algorithm 3.4: Approximate policy iteration: State-action value function approximation

```
$k \leftarrow 0$;
$q_k \leftarrow \mathbf{0}$ ;
while $\pi_k \neq \pi_{k-1}$ do
    $\pi_k(s) \leftarrow \arg\max_{a \in \mathcal{A}} q(s,a) \quad \forall s \in \mathcal{S}$ ;        // Greedy policy
    $q_k(\pi_k) \leftarrow \mathcal{Z}(\pi_k)$ ;
    $k \leftarrow k+1$;
return $q_k$;
```

3.2 Bounds on Approximation Error

This section describes upper and lower bounds on the error of approximate policy and value iteration. The lower bounds show that the approximate policy iteration guarantees cannot match the guarantees of approximate linear and bilinear programming.

All bounds in this chapter assume that all samples are present — essentially that the Bellman operator is known and the sampling error is 0. Deriving sampling bounds for the iterative algorithms is nontrivial and represents one of their drawbacks.

3.2.1 Upper Bounds

Much effort has been devoted to deriving bounds on the solution quality of approximate policy and value iteration. These bounds typically consider L_∞-API because the Bellman operator provides the simplest guarantees in terms of the L_∞ norm (Guestrin et al., 2003). The standard error bounds for approximate policy iteration follows.

Theorem 3.2 (e.g. (Bertsekas & Tsitsiklis, 1996) Chapter 2.6)**.** *Let $\hat{v}_k$ be the value function of the policy in step k in L_∞-API and let $\tilde{v}_k$ be the approximate value function. Then:*

$$\limsup_{k\to\infty} \|v^* - \hat{v}_k\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{k\to\infty} \|\tilde{v}_k - v_k\|_\infty.$$

This standard theorem, however, does not fit the simplified setting, in which the Bellman residual is minimized. We prove a different version of the theorem below.

Theorem 3.3. *Let $\hat{v}_k$ be the value function of the policy π_k in step k in L_∞-API. Then:*

$$\limsup_{k\to\infty} \|v^* - v_k\|_\infty \leq \frac{2\gamma}{(1-\gamma)^3} \limsup_{k\to\infty} \min_{v\in\Phi} \|L_{\pi_k} v - v\|_\infty.$$

The proof of the theorem can be found in Section C.4. This bound is probably loose by a factor $1/(1-\gamma)$ compared to some other bounds in the literature (Munos, 2003).

The bounds on solution quality of iterative algorithms have several undesirable properties. First, they require that the value functions of all policies $\pi_1, \pi_2, \ldots$ can be represented well using the features. This makes it hard to design appropriate features for a problem domain.

It is more desirable to bound the error in terms of a single value, such as the closest possible approximation of the optimal value function $\min_{v \in \mathcal{M}} \|v^* - v\|_\infty$. Such bounds are impossible to achieve as Section 3.2.2 shows.

Second, the bounds have multiplicative factors of at least $1/(1-\gamma)^2$. This constant may be very large when the discount factor γ is close to one. For example, discount factors of $\gamma = 0.999$ are common, in which case the bound may overestimate the true error by a factor of almost 10^6. These approximation bounds can be loose as we show below. It is possible to take advantage of the looseness of the bounds to improve the performance of approximate policy iteration on some problems (Petrik & Scherrer, 2008).

Finally, the bounds hold only in the limit. This is an inherent limitation of the iterative methods, which may not converge in general. As a result, there is no final solution that can be analyzed and bounded. The methods may converge under some restrictive assumptions as Section 3.3 shows.

We now show that the bounds on the approximation error bounds may be loose, in particular when $\gamma \to 1$. First, there exists a naive bound on the approximation error that can be dramatically tighter than the standard bounds when γ is close to 1.

Proposition 3.4. *Assume a policy $\pi \in \Pi$. There exists a constant $c \in \mathbb{R}$ such that for all $\gamma \in (0, 1)$:*

$$\|v^* - v_\pi\|_\infty \le \frac{c}{1-\gamma},$$

There exists an MDP such that for some $c' \in \mathbb{R}$, which is independent of γ:

$$\frac{1}{(1-\gamma)^2}\|v_\pi - Lv_\pi\|_\infty \ge \frac{c'}{(1-\gamma)^3}.$$

Here, the left-hand side represents the bound from Theorem 3.2.

The proof of the proposition can be found in Section C.4. This proposition shows that the bound is loose by a factor of at least $1/(1-\gamma)^2$. For example, in the MDP formulation of Blackjack (Parr, Li, Taylor, Painter-Wakefield, & Littman, 2008) the discount factor $\gamma = 0.999$, in which case the error bound may overestimate the true error by a factor up to $1/(1-\gamma)^2 = 10^6$.

Proposition 3.4 implies that for every MDP, there exists a discount factor γ, such that Theorem 3.2 is not tight. The looseness of the approximation error bounds may seem to contradict Example 6.4 in (Bertsekas & Tsitsiklis, 1996) that shows that Theorem 3.2 is tight. The discrepancy arises because we assume that the MDP has fixed rewards and number of states, while the example in (Bertsekas & Tsitsiklis, 1996) assumes that the reward depends on the discount factor and the number of states is infinite. Another way to put it is to say that Example 6.4 shows that for any discount factor γ there exists an MDP (which depends on γ) for which the bound Theorem 3.2 is tight. We, on the other hand, show that there does not exist a fixed MDP such that the bound Theorem 3.2 is tight for all discount factors γ.

3.2.2 Lower Bounds

As mentioned above, it is desirable to have bounds on the quality of the solution that do not require approximating policies $\pi_1, \pi_2, \ldots$. Unfortunately, bounds in terms of the closest approximation of the optimal value function v^* is impossible as the following theorem shows.

Theorem 3.5. *Let $\tilde{v}$ be a solution of approximate policy or value iteration taken at any iteration. Then, there exists* no *constant c (independent of the representation $\mathcal{M}$) such that:*

$$\|v^* - \tilde{v}\|_\infty \leq c \min_{v \in \mathcal{M}} \|v^* - v\|_\infty$$
$$\|\tilde{v} - L\tilde{v}\|_\infty \leq c \min_{v \in \mathcal{M}} \|v - Lv\|_\infty$$

This result applies to approximate policy iteration with both L_2 Bellman residual and least-squares minimizations. The bounds also apply when the iterative algorithms converge.

The proof of the theorem can be found in Section C.4. The theorem shows that approximate value and policy iterations may not converge to v^*, even if v^* is representable ($v^* \in \mathcal{M}$). Notice that this property differs from the standard divergence analysis of temporal difference (TD) learning (Bertsekas & Tsitsiklis, 1996; Baird, 1995). TD learning is a policy evaluation method that iteratively calculates a modified version of the least squares projection. Our analysis shows divergence in the policy optimization phase of the algorithms, not the policy evaluation one.

Theorem 3.5 applies to approximate policy iteration that minimizes the L_2 norm, which we chose because its popularity in practice. It is likely that similar bounds also apply to L_∞-API.

Although the iterative algorithms do not converge to the optimal value function even when it is representable, their performance may depend on the initialization, as the following proposition shows.

Proposition 3.6. *The optimal value function v^* is a fixed point of approximate policy and value iteration whenever it is representable ($v^* \in \mathcal{M}$).*

The proof of the proposition can be found in Section C.4. This proposition implies that when the iterative algorithms are initialized to the optimal value function they will also converge to it. The solution quality of the iterative algorithms may be, therefore, improved by choosing an appropriate initialization. Note, however, that these bounds assume that there is no sampling error.

3.3 Monotonous Approximation: Achieving Convergence

Section 3.2.1 show that the error bounds can be quite loose in many settings. We show now that it is possible to achieve tighter bounds under some restrictive assumptions on the approximation features and problem structure. We also show that, interestingly, the bounds can be tightened by using a discount factor that is different from γ. These results mildly generalize convergence results that can be obtained with averagers (Gordon, 1995) and the bounds on aggregation for MDPs (Bertsekas & Tsitsiklis, 1996).

Approximation error bounds can be tightened when the iterative algorithms converge. In particular, the bound in Theorem 3.2 can be strengthened to:

$$\|v^* - \hat{v}\|_\infty \leq \frac{2}{1-\gamma} \|L\tilde{v}_k - \tilde{v}_k\|_\infty,$$

where $\hat{v}$ is the solution of approximate policy iteration (see Theorem 2.15). This bound removes the dependence on the errors encountered during the optimization, but is not very helpful in analyzing the properties of the policy. The convergence however does not improve on Theorem 3.5 since in that case the approximate dynamic programming methods converge to the optimal solution.

While plain convergence does not significantly improve the solution quality of approximate policy iteration, it is possible to get convergence and better bounds in some special cases. For example, if the approximation operator $\mathcal{Z}$ is monotonous, the following theorem shows that the methods converge to a solution closest to the optimal value function within a multiplicative factor. This error bound is, in fact, close to properties of some of the optimization–based algorithms.

Theorem 3.7. *Let Z be a* monotonous *(see Definition C.1) approximation operator and assume that either least-squares approximate policy or value iterations converge to $\tilde{v}$. Then we have:*

$$(\mathbf{I} - ZP^*)^{-1}(Z - \mathbf{I})v^* \le \tilde{v} - v^* \le (\mathbf{I} - Z\tilde{P})^{-1}(Z - \mathbf{I})\tilde{v}^*$$
$$\|\tilde{v} - v^*\|_\infty \le \frac{1}{1-\gamma} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty,$$

where P^* and $\tilde{P}$ are the transition matrices of policies greedy with regard to v^* and v respectively.

The proof of the theorem can be found in Section C.4.

Monotonicity appears to be a reasonable property to be expected from the approximation operator. It ensures that when the true value function improves, the approximation does not become worse. Unfortunately, monotonicity is also quite limiting when used with linear approximation — the only linear approximation method, which is monotonous is aggregation.

Definition 3.8. A linear operator represents an *aggregation* if there exists a basis $\Phi = [\phi_1 \dots \phi_n]$ of the approximation space such that

$$\forall s \in \mathcal{S}\ \exists i \quad \phi_i(s) > 0 \ \Rightarrow \forall j \neq i \quad \phi_j(s) = 0.$$

Aggregation, in contrast with general approximation, firmly defines the ratios among the states in a single aggregate state. Each individual state belongs to exactly one aggregate state.

Theorem 3.9. *A linear approximation is monotonous if and only if it represents an aggregation.*

The proof of the theorem can be found in Section C.4. This theorem shows that while monotonous approximations may have good properties, they are in general very restrictive.

3.4 Contributions

This chapter focuses on the weaknesses of existing algorithms and shows some new, but simple, properties. In particular, the bound in Theorem 3.3 is very closely related to existing bounds and is proved using similar techniques. The lower bounds on the performance in

Theorem 3.5 are new, as well as the analysis that shows looseness of existing error bounds in Proposition 3.4. We are not aware of previous work that studies monotonous approximators, such as Theorem 3.7, but this is closely related to the notion of averagers (Gordon, 1995). Finally, Theorem 3.9 is new.

CHAPTER 4

APPROXIMATE LINEAR PROGRAMMING: TRACTABLE BUT LOOSE APPROXIMATION

This chapter describes the formulation and basic properties of approximate linear programming (ALP). ALP is a classical optimization–based method, which offers stronger guarantees and simpler analysis than the iterative methods. However, it often does not perform well in practice. In addition to basic properties, we show fundamental modifications of ALP that improve its theoretical and practical performance.

The chapter is organized as follows. First, Section 4.1 describes the basic approximate linear program formulations and the principles used to derive them. Then, Section 4.2 describes how to formulate approximate linear programs from samples and their properties. Section 4.3 shows offline approximation error bounds for approximate linear programs including sampling error bounds. Section 4.4 shows that the bounds, though better than iterative methods, are not sufficient to guarantee good solution quality. Sections 4.5 and 4.6 propose improvements to ALP that provide both tighter bounds and better performance. Finally, Section 4.7 shows empirical results and Section 4.8 discusses the practical implications.

4.1 Formulation

The formulation in this chapter slightly refines the traditional ALP formulation (de Farias, 2002). Approximate linear programming minimizes the expected policy loss in Definition 2.10 by minimizing the online approximation error bound in Theorem 2.19, which states that:

$$\|v^* - v_\pi\|_{1,\alpha} \le \left(\bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T}\right)(\tilde{v} - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ .$$

That means computing the approximate value function as follows:

$$\min_{\pi\in\Pi} \|v^* - v_\pi\|_{1,\alpha} \le \min_{v\in\mathcal{M}} \left(\bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T}\right)(\tilde{v} - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ .$$

This objective function is nonlinear, because of the term $\bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+$. While this term can be easily linearized, the standard approach it to restrict the search to transitive-feasible value functions. Then, as Theorem 2.19 shows:

$$\min_{\pi\in\Pi} \|v^* - v_\pi\|_{1,\alpha} \le \min_{v\in\mathcal{M}\cap\mathcal{K}} \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)(\tilde{v} - v^*)$$

An even simpler formulation, as Remark 2.18 shows, is:

$$\min_{\pi\in\Pi} \|v^* - v_\pi\|_{1,\alpha} \le \min_{v\in\mathcal{M}\cap\mathcal{K}} \bar{u}^\mathsf{T}(\tilde{v} - v^*)$$

This represents the following mathematical program:

$$\begin{aligned} \min_{v} \quad & c^{\mathsf{T}} v \\ \text{s.t.} \quad & v \in \mathcal{K} \\ & v \in \mathcal{M}(\psi) \end{aligned} \tag{ALP}$$

Here, $\mathcal{M}(\psi)$ represents the set of representable value functions as defined in Definition 2.20, and $c^{\mathsf{T}} = \bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*)$. Note that Remark 2.18 can be used to define an alternative — simpler and looser — formulation with $c^{\mathsf{T}} = \bar{u}^{\mathsf{T}}$. The remainder of the section now shows how to formulate $v \in \mathcal{K}$ as a set of linear constraints.

The formulation above relies on knowing a bound on the state-visitation frequencies of the solution policy $\bar{u}$ and the optimal transition matrix P^*; these values are typically unknown. Usually, heuristic choices are used instead. Because c can be scaled arbitrarily without affecting the result, in the remainder of the section we assume that it sums to one. That is:

$$c^{\mathsf{T}} \mathbf{1} = 1.$$

An alternative intuition for formulating the approximate linear program is that it is minimizing an upper bound on the optimal value function. Transitive-feasible value functions overestimate the optimal value function without actually using it in the definition, and the objective selects a minimal such function according to some linear measure.

A value function v is transitive-feasible if and only if it satisfies for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$:

$$v(s) \geq r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') v(s').$$

To show that such a function is transitive-feasible, let $s \in \mathcal{S}$ be an arbitrary state:

$$v(s) \geq r(s,a^*) + \gamma \sum_{s' \in \mathcal{S}} P(s,a^*,s') v(s') = \max_{a \in \mathcal{A}} r(s,a^*) + \gamma \sum_{s' \in \mathcal{S}} P(s,a^*,s') = (Lv)(s),$$

where a^* is the appropriate maximizer. When $\mathcal{M} = \mathbb{R}^{|\mathcal{S}|}$ — that is all value functions are representable — the approximate linear program solves $\min_{v \in \mathcal{K}} \|v^* - v\|_{1,c}$ and is formulated as:

$$\begin{aligned} \min_{v} \quad & \sum_{s \in \mathcal{S}} c(s) v(s) \\ \text{s.t.} \quad & v(s) \geq r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s',s,a) v(s') \quad \forall (s,a) \in (\mathcal{S}, \mathcal{A}) \end{aligned} \tag{4.1}$$

The number of constraints of the linear program is $|\mathcal{A}| \cdot |\mathcal{S}|$ and the number of variables is $|\mathcal{S}|$. This linear program is used to solve MDPs precisely as the following proposition shows.

Proposition 4.1 (e.g.(Puterman, 2005))**.** *The optimal value function v^* is an optimal solution of the linear program* (4.1) *whenever $c(s) > 0$ for all $s \in \mathcal{S}$.*

When the value function is restricted to be representable using a basis matrix Φ, the approximate liner program becomes:

$$\begin{aligned}
\min_{x_1 \ldots x_m} \quad & \sum_{s \in \mathcal{S}} \sum_{i=1}^{m} c(s) \phi(s')_i x_i \\
\text{s.t.} \quad & v(s) \geq r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s', s, a) \sum_{i=1}^{m} \phi(s')_i x_i \quad \forall (s,a) \in (\mathcal{S}, \mathcal{A})
\end{aligned} \tag{4.2}$$

Here, m is the number of features ϕ_i. Notice that the number of variables of the linear program corresponds to the number of features. When the set of representable features is regularized, there is an additional constraint: $\|x\|_{1,e} \leq \psi$.

The linear program (4.2) can be expressed in matrix form. Let A be the constraint matrix and b the right-hand side.

$$A = \begin{pmatrix} \mathbf{I} - \gamma P_{a_1} \\ \mathbf{I} - \gamma P_{a_2} \\ \vdots \end{pmatrix} \qquad b = \begin{pmatrix} r_{a_1} \\ r_{a_2} \\ \vdots \end{pmatrix}.$$

The approximate linear program representations in matrix form is:

$$\begin{aligned}
\min_{x} \quad & c^{\mathsf{T}} \Phi x \\
\text{s.t.} \quad & A \Phi x \geq b
\end{aligned}$$

When regularization is used, the additional constraint is $\|x\|_{1,e} \leq \psi$. We discuss in more detail how to formulate this constraint linearly in Chapter 6.

Approximate linear programs can be also formulated based on state-actions values q instead of simple value functions. State–action formulation simplifies the calculation of a greedy policy in some complex domains, when calculating it requires solving an optimization problem. When only a value function is available, calculating the greedy action entails:

$$\max_{a \in \mathcal{A}} r(s,a) + \gamma \mathbf{E}\left[v(S)\right],$$

where the expectation is over the next state S using the transition probabilities. This significantly complicates the computation. On the other hand, when the state–action value function is available, the greedy action is computed as:

$$\max_{a \in \mathcal{A}} q(s,a),$$

avoiding the expectation.

The state–action based ALP is closely related to the linear program (ALP), using the fact that $v(s) = \max_{a \in \mathcal{A}} q(s,a)$. The set of feasible state–action value functions satisfy:

$$q_a \geq r_a + \gamma P_a q_{a'} \qquad \forall a, a' \in \mathcal{A}.$$

The representable state–action value functions in the ALP can use different features for all actions of each state.

The actual linear program formulation is then the following.

$$\begin{aligned} \min_{v,q} \quad & c^{\mathsf{T}} v \\ \text{s.t.} \quad & v \geq q_a \qquad \forall a \in \mathcal{A} \\ & q_a \geq \gamma P_a v + r_a \\ & q \in \mathcal{M} \end{aligned} \tag{4.3}$$

The value function v need not be representable. This is because in the case of sampled constraints, the number of states for which v needs to be defined corresponds only to the number of constraints in the linear program. The number of constraints must be small, as the next section shows.

4.2 Sample-based Formulation

This section discusses how to formulate an approximate linear program from state samples. There are two main reasons why the approximate linear program needs to be formulated based on samples. First, in some cases the full description of the domain is not available, and only examples of states transitions can be used. Second, even when the full description is available, it is usually impossible to even write down the linear program. The number of constraints corresponds to the number of states and actions in the domain, which can be very large. Samples make it easier to express and solve the linear program.

There are two main formulations, depending on the type of samples that are available. The sample types and relevant assumptions on their properties are described in Section 2.4.2.

Sampled ALP In this formulation, the samples are used to construct the rows of the transition matrix. In comparison to the full formulation in (ALP), the linear program is missing some constraints. This is expressed by constraining the value functions to be transitive feasible with respect to the sampled Bellman operator as follows:

$$\begin{aligned} \min_{v} \quad & \bar{c}^{\mathsf{T}} v = \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s} \left(\sum_{s \in \bar{\Sigma}} c(s) \mathbf{1}_s \right)^{\mathsf{T}} v \\ \text{s.t.} \quad & v \in \bar{\mathcal{K}} \\ & v \in \mathcal{M}(\psi) \end{aligned} \tag{s–ALP}$$

The formulation in terms of matrices is:

$$\begin{aligned} \min_{x} \quad & \bar{c}^{\mathsf{T}} \Phi x \\ \text{s.t.} \quad & \bar{A} \Phi x \geq \bar{b} \end{aligned}$$

where for all $(s_i, a_j) \in \bar{\Sigma}$.

$$\bar{A}\Phi = \begin{pmatrix} - & \phi(s_i)^\mathsf{T} - \gamma \sum_{s' \in \mathcal{S}} P(s_i, a_j, s')\phi(s')^\mathsf{T} & - \\ - & \vdots & - \end{pmatrix}$$

$$\bar{b} = \begin{pmatrix} r(s_i, a_j) \\ \vdots \end{pmatrix}$$

$$\bar{c}^\mathsf{T}\Phi = \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s} \sum_{s \in \bar{\Sigma}} c(s) \mathbf{1}_s^\mathsf{T} \Phi = \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s} \sum_{s \in \bar{\Sigma}} c(s)\phi(s)^\mathsf{T}$$

The value $\bar{c}$ as defined above does not necessarily sum to one. This is, however, not an issue since scaling the objective function does not influence the optimal solution. The constant $\frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}$ is also not influence the solution and serves only to ensure that the scale of $\bar{c}$ is the same as the scale of c. Figure 4.1 shows how the sampling reduces the number of constraints. It shows that the number of constraints in the linear program (s–ALP) corresponds to $|\Sigma|_a$ instead of $|\mathcal{S}| \cdot |\mathcal{A}|$. This makes the linear program easier to solve.

Estimated ALP In this formulation, the samples are used to construct the rows of the transition matrix. In comparison to the full formulation in (ALP), the linear program is missing some constraints *and* the constraints are imprecise. That is, the number of constraints is the same as in the sampled formulation, but the constraints are potentially imprecise. This is expressed by constraining the value functions to be transitive feasible with respect to the estimated Bellman operator as follows:

$$\begin{aligned} \min_{v} \quad & \tilde{c}^\mathsf{T} v = \frac{|\mathcal{S}|}{|\tilde{\Sigma}|_s} \left(\sum_{s \in \tilde{\Sigma}} c(s) \mathbf{1}_s \right)^\mathsf{T} v \\ \text{s.t.} \quad & v \in \tilde{\mathcal{K}} \\ & v \in \mathcal{M}(\psi) \end{aligned} \tag{e–ALP}$$

The formulation in terms of matrices is:

$$\begin{aligned} \min_{x} \quad & \tilde{c}^\mathsf{T}\Phi x \\ \text{s.t.} \quad & \tilde{A}\Phi x \geq \tilde{b} \end{aligned}$$

where for all $(s_i, a_j) \in \tilde{\Sigma}$.

$$\tilde{A}\Phi = \begin{pmatrix} - & \phi(s_i)^\mathsf{T} - \gamma \frac{1}{m} \sum_{s' \in s'_1 \dots s'_m} P(s_i, a_j, s')\phi(s')^\mathsf{T} & - \\ - & \vdots & - \end{pmatrix}$$

$$\tilde{b} = \begin{pmatrix} r(s_i, a_j) \\ \vdots \end{pmatrix}$$

$$\tilde{c}^\mathsf{T}\Phi = \frac{|\mathcal{S}|}{|\tilde{\Sigma}|_s} \sum_{s \in \tilde{\Sigma}} c(s) \mathbf{1}_s^\mathsf{T} \Phi = \frac{|\mathcal{S}|}{|\tilde{\Sigma}|_s} \sum_{s \in \tilde{\Sigma}} c(s)\phi(s)^\mathsf{T}$$

The value $\tilde{c}$ as defined above does not necessarily sum to one. This is, however, not an issue since scaling the objective function does not influence the optimal solution. This formulation is very similar to sampled ALP. The main difference is that the rows of the constraint matrix $\tilde{A}$ are not known precisely, but are only estimated. The number of constraints in the linear program (e–ALP) also corresponds to $|\Sigma|_a$ instead of $|\mathcal{S}| \cdot |\mathcal{A}|$. This makes the linear program easier to solve.

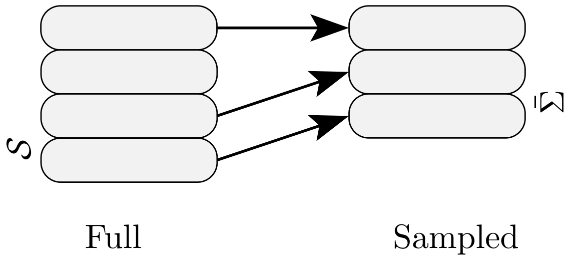

Figure 4.1. Sampled constraint matrix A of the approximate linear program.

4.3 Offline Error Bounds

This section discusses the offline bounds on solutions of approximate linear programs. Offline error bounds — described in more detail in Section 2.5 — determine the quality guarantees of the value function approximation algorithm. The description of the formulation above trivially shows the following proposition.

Theorem 4.2 (Offline Policy Loss)**.** *Assume Assumption 2.21 and let $\tilde{v}$ be the solution of the approximate linear program in* (ALP) *and π be the greedy policy. Then:*

$$\|v^* - v_\pi\|_{1,\alpha} \leq \min_{v \in \mathcal{M} \cap \mathcal{K}} \bar{u}^\mathsf{T} (\mathbf{I} - \gamma P^*)(v - v^*) \leq 2 \bar{u}^\mathsf{T} \mathbf{1} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty$$

when $c^\mathsf{T} = \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)$. In addition,

$$\|v^* - v_\pi\|_{1,\alpha} \leq \min_{v \in \mathcal{M} \cap \mathcal{K}} \bar{u}^\mathsf{T} (v - v^*) \leq \frac{2 \bar{u}^\mathsf{T} \mathbf{1}}{1 - \gamma} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty,$$

when $c^\mathsf{T} = \bar{u}^\mathsf{T}$. Notice that the first bound is tighter by a factor $1 - \gamma$.

The proof of the theorem can be found in Section C.5.

For comparison with other bounds, note that $\bar{u}^\mathsf{T} \mathbf{1} \geq 1/(1 - \gamma)$ (see Lemma C.10). The bound above is not practical because it relies on $\bar{u}$, which is often unavailable and cannot be easily estimated. Because of that, ALP offline bounds typically focus on the approximate value function instead of the policy. That is, the bounds are on $\|v^* - \tilde{v}\|$ instead of $\|v^* - v_\pi\|$ for some norm. Albeit this makes the bounds somewhat arbitrary, they are much easier to analyze. The following is a standard approximation error bound.

Theorem 4.3 (e.g. (de Farias, 2002))**.** *Assume Assumption 2.21 and let $\tilde{v}$ be the solution of the approximate linear program* (ALP)*.*

$$\|v^* - \tilde{v}\|_{1,c} \leq \frac{2}{1 - \gamma} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty.$$

The theorem follows simply from the properties of approximate linear programs, like Theorem 4.2. The first part of the theorem is a specialization of Theorem 4.4, which is described below.

The bounds presented above are all for the full linear program in (ALP) and do not account for incomplete sets of samples. A generalized version, which accounts for sampling follows. Because of the difficulties with bounding the policy loss, we focus only on bounding $\|v - v^*\|_{1,c}$.

Theorem 4.4. *Assume Assumptions 2.21, 2.26, 2.27, 2.28 and let v_1, v_2, v_3 be the optimal solutions of* (ALP), (s–ALP), *and* (e–ALP) *respectively. Then, the following inequalities hold:*

$$\begin{aligned}
\|v_1 - v^*\|_{1,c} &\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty \\
\|v_2 - v^*\|_{1,c} &\le \|v_1 - v^*\|_{1,c} + 2\frac{\epsilon_p(\psi)}{1-\gamma} \\
&\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty + 2\epsilon_c(\psi) + 2\frac{\epsilon_p(\psi)}{1-\gamma} \\
\|v_3 - v^*\|_{1,c} &\le \|v_1 - v^*\|_{1,c} + 2\epsilon_c(\psi) + \frac{3\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma} \\
&\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty + 2\epsilon_c(\psi) + \frac{3\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma}
\end{aligned}$$

The last bound can be tightened to $2\epsilon_s$ from $3\epsilon_s(\psi)$ when $\epsilon_c = 0$.

The proof of the theorem can be found in Section C.5.

The offline error bounds in Theorem 4.4 can be used to guarantee the performance of an ALP for a fixed number of samples and the regularization coefficient ψ. It does not, however, prescribe how to choose the regularization coefficient for a given set of samples. To do that, we have to derive bounds for an actual value function v. When the samples are known, these bounds are typically tighter than the offline error bound.

Theorem 4.5 (Online Error Bound)**.** *Let $\tilde{v} \in \tilde{\mathcal{K}} \cap \mathcal{M}$ be an arbitrary feasible solution of the estimated ALP* (e–ALP)*. Then:*

$$\|v^* - \tilde{v}\|_{1,c} \le \tilde{c}^\mathsf{T}\tilde{v} - c^\mathsf{T} v^* + \frac{2\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma}$$

The proof of the theorem can be found in Section C.5. Notice the missing constant ϵ_c when compared to Theorem 4.4. This is because the bound involves $\tilde{c}^\mathsf{T} v$, not $c^\mathsf{T} v$.

4.4 Practical Performance and Lower Bounds

While the ALP formulation offers a number of favorable theoretical properties over iterative algorithms, it often under-performs in practice. The reason why ALP under-performs can be traced to the approximation error bounds. The constants in the ALP performance bounds (Theorem 4.2 and Theorem 4.4), is typically $1/(1-\gamma)$. While this is more favorable than the constant $1/(1-\gamma)^2$ in iterative algorithms (Theorem 3.2), this number can be very high for discount factors close to 1. As we show below, the constant $1/(1-\gamma)$ in the bounds is truly necessary and the bound may be tight in some problems.

Assuming that an upper bound on the policy visitation frequencies $\bar{u}$ is available, approximate linear program minimizes bounds in Theorem 2.19 or Remark 2.18 instead of Theorem 2.17. In particular, instead of optimizing:

$$\|v^* - v_\pi\|_{1,\alpha} \leq -\|v^* - \tilde{v}\|_{1,\alpha} + \|\tilde{v} - L\tilde{v}\|_{1,\bar{u}}$$

ALP, in its simplest incarnation, optimizes:

$$\|v^* - v_\pi\|_{1,\alpha} \leq \|v^* - \tilde{v}\|_{1,\bar{u}}.$$

These ALP bounds may be quite loose, as the following proposition shows.

Proposition 4.6. *For any $\tilde{v} \in \mathcal{K}$ there exists no constant $c \in \mathbb{R}$ such that:*

$$\bar{u}^\mathsf{T}(\tilde{v} - v^*) \leq c\bar{u}^\mathsf{T}(\tilde{v} - L\tilde{v}),$$

even when Assumption 2.21 is satisfied. This holds for the precise Bellman operator L, not assuming sampling. In addition, for all $\tilde{v} \in \mathcal{K}$:

$$(\tilde{v} - v^*) \geq (\tilde{v} - L\tilde{v}).$$

The proof of the proposition can be found in Section C.5. The tighter bounds in Theorem 2.17 that use the Bellman residual can be minimized by approximate bilinear programming, described in Chapter 5.

This demonstrates that ALP is minimizing loose bounds. The following simple example illustrates when approximate linear program fails to approximate the value function well. Consider the simple deterministic chain problem with a discount factor $\gamma = 0.9$, depicted in Figure 4.2. There is a single action, one constant feature ϕ_1, and a feature ϕ_2 that corresponds to the optimal value function with the exception that $\phi_2(s_5) = \phi_2(s_6)$.

The optimal value function v^*, the closest approximation v_1 in terms of L_∞ norm, and the solution v_2 of an ALP are depicted in Figure 4.3. It is apparent that the approximation error in this problem is too large to make the value useful. This example shows the following, when assuming that $c = \alpha = \mathbf{1}_{s_7}$:

Proposition 4.7. *There exists an MDP such that for the optimal solution $\tilde{v}$ of* (ALP) *we have that*

$$\|\tilde{v} - v^*\|_{1,\alpha} \geq \frac{2}{1-\gamma} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

This holds even when Assumption 2.21 is satisfied. In addition, if $c = u_\pi$ (which is unknown), the ALP bound on the policy loss is:

$$\|\tilde{v} - v^*\|_{1,u_\pi} \geq \frac{2}{(1-\gamma)^2} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

There also exists an MDP such that for the greedy policy π with respect to the ALP solution the policy loss is:

$$\|v^* - v_\pi\|_{1,\alpha} \geq \frac{2\gamma}{1-\gamma} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

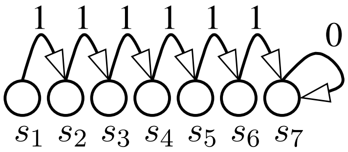

Figure 4.2. An example chain problem with deterministic transitions and reward denoted above the transitions.

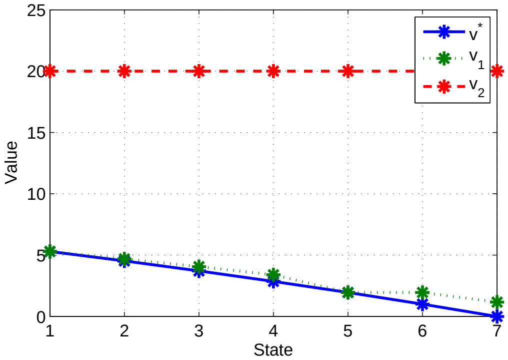

Figure 4.3. Approximation errors in approximate linear programming.

The proof of the proposition can be found in Section C.5. This proposition indicates that the online error bound minimized by approximate linear programming is very loose. In particular, when the discount factor γ is close to one, the term $1/(1-\gamma)^2$ grows very rapidly. The guarantees provided by approximate linear programming are, as a result, very loose. Note that the offline error bound on the policy loss is almost as loose as possible, as Proposition 3.4 show (it cannot be a function of $1/(1-\gamma)^2$).

While it is possible to reduce the transitional error by requiring that an appropriate structure is present in the basis, this is not always practical. An example of such structure is the Lyapunov vectors (de Farias, 2002). The existence of such vectors in all but the simplest problem domains is typically hard to ensure.

The results of a direct application of ALP to blood inventory management are similarly disappointing, as Section 4.7 describes in more detail. The solution is significantly worse than a simple myopic optimization, which only optimizes over a single step and does not consider the future. The solution of the ALP overestimates the value of the inventory and does not dispense any blood.

The poor performance of ALP in the blood inventory management problem is due to the fact that the solution significantly overestimates the value of the resources. While the optimal value function in this problem is not known, we hypothesize that the function is concave. When approximating a concave function by a linear one, the ALP formulation can be seen as approximating the upper bound on the functions derivative. This leads to a large approximation error, as demonstrated in Figure 4.4.

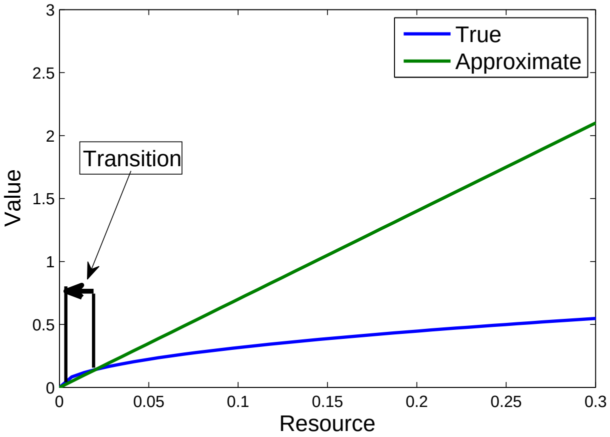

Figure 4.4. Illustration of the value function in the inventory management problem.

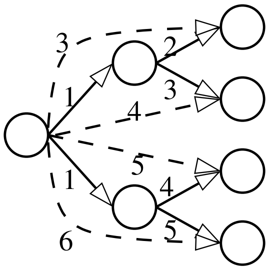

Figure 4.5. An example of 2-step expanded constraints (dashed) in a deterministic problem. The numbers next to the arcs represent rewards.

4.5 Expanding Constraints

The algorithmic error would be zero, if the approximate linear program constraints were in the form $v(s) \geq v^*(s)$ instead of having a constraint for each transition. This is however quite impractical even in very small problems. Using direct constraints results in many potential constraints per state. This would dramatically increase the number of sampled constraints needed to guarantee a small state sampling error. We therefore consider a hybrid formulation, in which the constraints represent multi-step transitions; we call these constraints *expanded*. An example of 2-step expanded constraints is depicted in Figure 4.5.

Definition 4.8. A t-step *expanded* ALP constraint for a *deterministic* MDP for an action sequence $E = (a_1, a_2, \ldots, a_t) \in \mathcal{A}^t$ and state s_i has the following form:

$$v(s_i) \geq \sum_{s_j \in \mathcal{S}} \gamma^t P^E(s_j \,|\, s_i)\, v(s_j) + r^E(s_i).$$

The terms are defined as follows, using matrix notation for simplicity:

$$P^E(s_j \mid s_i) = \mathbf{1}_i^{\mathsf{T}} \left(\prod_{k=1}^{t} \gamma P_{a_k} \right) \mathbf{1}_j$$

$$r^E(s_i, E) = \mathbf{1}_i^{\mathsf{T}} \left(\sum_{l=1}^{t} \left(\prod_{k=1}^{l-1} \gamma P_{a_l} \right) r_{a_k} \right),$$

where $\mathbf{1}_i$ is a i-th unit vector. We denote such a constraint as $v(s_i) \geq \mathcal{C}(s_i, E)$. A *full* t-step *expanded* constraint is:

$$v(s_i) \geq \max_{E \in \mathcal{A}^t} \mathcal{C}(s_i, E).$$

Note that constraints can be expanded in the way defined above only for deterministic MDPs, which are defined as follows.

Definition 4.9. A Markov decision process is deterministic when for every $s \in \mathcal{S}$ and every action $a \in \mathcal{A}$, there exists an $s' \in \mathcal{S}$ such that $P(s, a, s') = 1$.

The definition can be easily extended to stochastic MDPs by considering policies instead of action sequences and is straightforward.

To simplify the analysis, we consider only *full* expanded constraints. That is, for every state we have constraints that correspond to all actions. This can be done by simply writing the max operator in terms of multiple constraints. For example, $a \geq \max\{b, c\}$ may be written as two constraints $a \geq b$ and $a \geq c$. Next we describe the benefits of using expanded constraints and we address the difficulties later. The set of constraints for all states is denoted as $\mathcal{C}_t$

This idea is somewhat related to temporally extended actions, or *options*, commonly studied in reinforcement learning (Stolle & Precup, 2002). Options, however, serve a different purpose – to simplify and accelerate learning rather than reduce the algorithmic error. In addition, unlike options, the expanded constraints have a fixed length.

The approximate linear program with expanded constraints is formulated as follows.

$$\begin{aligned} \min_{v} \quad & c^{\mathsf{T}} v \\ \text{s.t.} \quad & v \geq \gamma^t P^E v + r^E \quad \forall E \in \mathcal{A}^t \\ & v \in \mathcal{M} \end{aligned} \tag{ALP–e}$$

In solving realistic problems, the linear program is also based on a sampled set of all the constraints. We do not analyze that case because its properties are very similar to regular approximate linear programs with sampled constraints.

The analysis of the approximate linear programs with expanded constraints relies on the following important property.

Lemma 4.10. *Let $\tilde{v}$ be feasible in* (ALP–e). *Then the following holds:*

$$\tilde{v} \geq L^t \tilde{v}$$
$$\tilde{v} \geq v^*,$$

where L^t represents t consecutive applications of the Bellman operator L. In addition, for any value function v such that $\|v - v^*\|_\infty \leq \epsilon$ there exists a v' feasible in (ALP–e) *defined as:*

$$v' = v + \frac{\epsilon}{1 - \gamma^t} \mathbf{1}.$$

The proof of the lemma can be found in Section C.5.

Expanding the constraints with a larger horizon guarantees improvement in the approximation error bounds. These offline approximation bounds from Theorem 4.3 can be extended using Lemma 4.10 as follows

Proposition 4.11. *Assume Assumption 2.21 and let $\tilde{v}$ be the solution of the t-step expanded approximate linear program* (ALP–e)*.*

$$\|v^* - \tilde{v}\|_{1,c} \leq \frac{2}{1-\gamma^t} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty.$$

This bound does not guarantee a reduction of the policy loss of the greedy policy in Theorem 4.2.

The proof of the proposition can be found in Section C.5. In fact the approximation error is guaranteed to not increase with increased t.

Corollary 4.12. *Let v be a solution of an ALP with t-step expanded constraints and let v' be a solution of an ALP with t'-step expanded constraints such that $t' = dt$ for some positive $d \in \mathbb{N}_+$. Then:*

$$\|v' - v^*\|_{1,c} \leq \|v - v^*\|_{1,c}.$$

The corollary follows trivially from the fact that $v \geq (L^t)^d v$. Note that this does not guarantee an improvement in the actual policy loss.

Expanding all constraints may improve the solution quality, but at a steep computational cost. The number of constraints required per states scales exponentially with the number of steps for which the constraints are expanded. As a result, assuming that full constraints are added for many steps into the future is not realistic if it needs to be done for all states. In this section, we propose a scheme for selecting only some constraints for expansion.

To obtain the bounds on improvement achievable by expanding constraints, we compare solutions of an ALP with and without some constraints expanded. Let $A_1 v \geq b_1$ represent constraints that are *not* expanded and let $A_2 v \geq b_2$ be the expanded constraints. Then, let $\bar{A}_1 v \geq \bar{b}_1$ be a fully expanded version of the constraint $A_1 v \geq b_1$. This leads to two linear programs:

$$\begin{aligned} \min_v \quad & c^\mathsf{T} v \\ \text{s.t.} \quad & A_1 v \geq b_1 \\ & A_2 v \geq b_2 \\ & v \in \mathcal{M} \end{aligned} \tag{4.4}$$

and

$$\begin{aligned} \min_v \quad & c^\mathsf{T} v \\ \text{s.t.} \quad & \bar{A}_1 v \geq \bar{b}_1 \\ & A_2 v \geq b_2 \\ & v \in \mathcal{M} \end{aligned} \tag{4.5}$$

We can now state the following proposition.

Proposition 4.13. *Let* v_1 *be the* optimal *solution of* (4.4) *and let* $\bar{v}_1$ *be the optimal solution of* (4.5)*. Let* λ_1 *and* λ_2 *be the Lagrange multipliers that correspond to constraints* $A_1 v \geq b_1$ *and* $A_2 v \geq b_2$ *respectively. Then the bound on the improvement from expanding constraints* $A_1 v \geq b_1$ *is at most:*

$$\begin{aligned}\|v_1 - v^*\|_{1,c} - \|\bar{v}_1 - v^*\|_{1,c} &\leq \|\lambda_1^\mathsf{T} A_1\|_1 \|v_1 - v_2\|_\infty \\ &\leq \frac{\|\left[A v_1 - b_1\right]_+\|_\infty}{1-\gamma} \|\lambda_1^\mathsf{T} A_1\|_1.\end{aligned}$$

The proof of the proposition can be found in Section C.5.

The proposition shows that the dual variables λ (or Lagrange multipliers) may be used to bound the potential improvement in the approximation that can be gained from expanding some of the constraints. In the trivial case, the proposition states that expanding constraints for which $\lambda = 0$ has no effect on the solution. Thus, it is sufficient to obtain a solution of the linear program in which the sum of the Lagrange multipliers of unexpanded constraints is sufficiently small. A simple greedy algorithm that accomplishes this is depicted in Algorithm 4.1. Note that after expanding a constraint, the previous solution of the ALP may be used as the starting point. Since this solution is feasible (see Corollary 4.12) and is likely close to the optimum, resolving the ALP is typically very easy.

Algorithm 4.1: Iterative Expansion Algorithm

```
while $\|v_1 - v^*\|_{1,c} - \|\bar{v}_1 - v^*\|_{1,c} \geq \epsilon$ do
    $v \leftarrow \arg\min_{v \in \mathcal{M}} c^\mathsf{T} v$ ;
    $\lambda_i \leftarrow$ dual variables ;
    $a_1 \ldots a_d \leftarrow \arg\max_{a_i} \|\lambda_i a_i\|_1$ ;
    Expand constraints $a_1 \ldots a_d$;
return $v$
```

We showed in this section a practical approach for obtaining expanded constraints in stochastic domains, as well as a method for selecting a subset of constraints to expand. In the next section we describe an approach that can also address the problem with virtual loops without requiring expansion of the constraints.

4.6 Relaxing Constraints

In this section, we describe a method for eliminating the virtual loops without requiring constraint expansion. This method is generally inferior to constraint expansion in terms of solution quality, since it cannot use the extra information. However, it is useful in problems in which rolling out constraints is not feasible and it is possible to obtain a good solution while violating only constraints with small weight.

This formulation is also based on minimization of the online error bounds. Recall that the standard linear program (ALP) minimizes (from Remark 2.18):

$$\min_{\pi\in\Pi} \|v^* - v_\pi\|_{1,\alpha} \leq \min_{v\in\mathcal{M}\cap\mathcal{K}} \bar{u}^\mathsf{T}(\tilde{v} - v^*),$$

while restricting the value functions to be transitive-feasible. Propositions 4.6 and 4.7 show that this bound can be particularly loose. We, therefore, propose to instead optimize the error bound from Theorem 2.19:

$$\min_{\pi\in\Pi} \|v^* - v_\pi\|_{1,\alpha} \leq \min_{v\in\mathcal{M}} \left(\bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T}\right)(\tilde{v} - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+,$$

which does not require that the value function is transitive feasible. This objective function is nonlinear, because of the term $\bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+$, but this can be easily linearized.

$$\begin{aligned}
\left[L\tilde{v} - \tilde{v}\right]_+(s) &= \left[\max_{a\in\mathcal{A}} \gamma P(s,a)^\mathsf{T} v + r(s,a) - v(s)\right]_+ \\
&= \left[\max_{\{a \,|\, \mathbf{1}^\mathsf{T}=1, a\geq\mathbf{0}\}} \gamma P(s,a)^\mathsf{T} v + r(s,a) - v(s)\right]_+ \\
&= \left[\min_{\{\lambda \,|\, \lambda\geq\gamma P(s,a(i))v + r(s,a_i) - v(s),\ \forall i=1\ldots|\mathcal{A}|\}} \lambda\right]_+
\end{aligned}$$

Here, we treat a as a vector and take the dual of the linear program. Note that $\bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_-$ cannot be linearized as easily. The linear program is then:

$$\begin{aligned}
\min_{v,\lambda} \quad & c^\mathsf{T} v + \sum_{s\in\mathcal{S}} \bar{u}(s)\lambda(s) \\
\text{s.t.} \quad & \sum_{s\in\mathcal{S}} \bar{\mathbf{1}}_s \lambda_s \geq b - Av \\
& \lambda_s \geq \mathbf{0} \\
& v \in \mathcal{M}
\end{aligned} \tag{ALP-r}$$

where:

$$\begin{aligned}
c^\mathsf{T} &= \left(\bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T}\right) \\
\bar{\mathbf{1}}_s\ (s', a') &= \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}.
\end{aligned}$$

Intuitively, this formulation is motivated by the simple chain example, described in Section 4.4. It shows that a single constraint in an ALP may degrade the quality of the whole solution because it forces transitive-feasible value functions to be very large. This can be addressed by allowing a small violation of a small number of constraints. One possibility of relaxing the constraints is to use: $Av \geq b - \epsilon\mathbf{1}$ for some small ϵ. This will however only lower the value function with little effect on quality.

Another motivation for (ALP-r) is its dual linear program, which is:

$$\begin{aligned}\max_{u} \quad & b^{\mathsf{T}} u \\ \text{s.t.} \quad & \Phi^{\mathsf{T}} A^{\mathsf{T}} u = \Phi^{\mathsf{T}} c \\ & u \geq \mathbf{0} \\ & \bar{\mathbf{1}}_s^{\mathsf{T}} u \leq \bar{u}(s)\end{aligned}$$

Except for the last constraint, this is identical to the linear program (C.2) — the dual of the linear program used to solve MDPs directly. The variables u correspond to the state–action visitation frequencies and the vector d then correspond to upper bounds on the dual values u. This is exactly $\bar{u}$ in our definition. The relaxation approach may be, therefore, seen as a regularization of the dual variables.

We can now show the following bound.

Theorem 4.14 (Offline Policy Loss)**.** *Let $\tilde{v}$ be the solution of the approximate linear program in* (ALP-r) *and π be the greedy policy. Then:*

$$\begin{aligned}\|v^* - v_\pi\|_{1,\alpha} &\leq \min_{v \in \mathcal{M}} \left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)(\tilde{v} - v^*) + \bar{u}^{\mathsf{T}}\left[L\tilde{v} - \tilde{v}\right]_+ \\ &\leq \left(\mathbf{1}^{\mathsf{T}}\bar{u}(1-\gamma) - 1 + 2(1+\gamma)\mathbf{1}^{\mathsf{T}}\bar{u}\right)\min_{v \in \mathcal{M}} \|v - v^*\|_\infty\end{aligned}$$

when $c^{\mathsf{T}} = \bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*)$ and the second inequality holds when $u^{\mathsf{T}}(\mathbf{I} - \gamma P^*) \geq \alpha^{\mathsf{T}}$. Note that the bound does not require Assumption 2.21 and does not involve transitive-feasible functions.

The proof of the theorem can be found in Section C.5.

While the linear program formulation (ALP-r) provides guarantees, it is relies on constants that are not known and are hard to estimate. In the remainder of the section, we study the properties of the following generic problem.

$$\begin{aligned}\min_{v,\lambda} \quad & c^{\mathsf{T}} v + d^{\mathsf{T}} \lambda \\ \text{s.t.} \quad & \lambda \geq b - Av \\ & \lambda \geq \mathbf{0} \\ & v \in \mathcal{M}\end{aligned} \tag{ALP-rx}$$

In this formulation, we assume that $\mathbf{1}^{\mathsf{T}} c = 1$ and d is an arbitrary value. We study the properties of (ALP-rx) with respect to the choice of d. To achieve a good performance of this formulation, it is necessary to select good values of the vector d. The values d can represent bounds on the dual solution, as the dual formulation of (ALP-rx) shows.

$$\begin{aligned}\max_{u} \quad & b^{\mathsf{T}} u \\ \text{s.t.} \quad & \Phi^{\mathsf{T}} A^{\mathsf{T}} u = \Phi^{\mathsf{T}} c \\ & u \geq \mathbf{0} \\ & u \leq d\end{aligned}$$

An important property of the linear program (ALP-rx) is that given a sufficiently large vector d, if the optimal value function v^* is representable in the approximation space d, then it will also be the optimal solution of the relaxed linear program.

Proposition 4.15. *Assume Assumption 2.21 and that*

$$d > \frac{\mathbf{1}^\mathsf{T} c}{1-\gamma}\mathbf{1}.$$

Then the sets of optimal solutions of (ALP) *and* (ALP-rx) *are identical.*

The proof of the proposition can be found in Section C.5.

The online approximation bounds above indicate that the values d should be upper bounds on the state visitation frequencies u in the MDP. One way of obtaining the values is to treat them as a parameter and use the value that minimizes the policy loss. This is practical in problems in which the solution time is dominated by the time required to gather the samples, and the time to solve the ALP is small. Clearly, this is not a satisfactory method.

The relaxed linear program formulation can be used when violating a small number of constraints significantly improves the solution. The formulation then automatically selects the appropriate constraints that need to be violated. The following proposition shows a bound on the total weight of the violated constraints.

Proposition 4.16. *Assume Assumption 2.21 and let I_V be the set of violated constraints and let I_A be the set of active constraints by the optimal solution of the relaxed approximate linear program. Then:*

$$d(I_V) \le \frac{\mathbf{1}^\mathsf{T} c}{1-\gamma} \le d(I_A) + d(I_V),$$

where $d(\cdot)$ denotes the sum of the weights defined by d on the selected constraints.

The proof of the proposition is very similar to the proof of Proposition 4.15.

The proposition implies that setting

$$d > \frac{\mathbf{1}^\mathsf{T} c}{(k+1)(1-\gamma)}\mathbf{1}$$

guarantees that at most k constraints are violated. The relaxation does not guarantee an improvement on the ALP solution. It is possible to construct an instance in which the solution can be improved only by violating all constraints.

In many problems, it may be possible to use the structure to derive bounds on the state–action violation frequencies $\bar{u}$. For example, if it is impossible to return to the same state s of the MDP in less than k steps, then $\bar{u}(s) \le 1/(1-\gamma^k)$. When the probability of returning is small in the first few steps, the following bound can be used.

Definition 4.17. The bound on the return probability $p_j(s)$ after k steps for any state $s \in \mathcal{S}$ is defined as:

$$p_k(s) \ge \max_{\pi\in\Pi} \mathbf{1}_s^\mathsf{T} P_\pi^k \mathbf{1}_s.$$

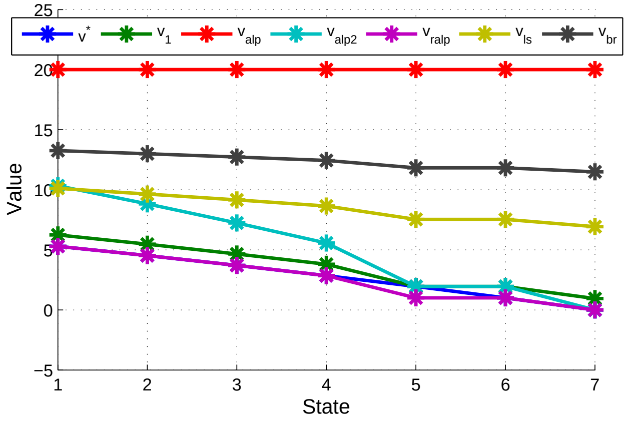

Figure 4.6. Benchmark results for the chain problem.

Proposition 4.18. *Let p_k be the probability of returning for the first time to state s. Then:*

$$u(s) \le \sum_{k=1}^{\infty} \gamma^k p_k(s) \le \sum_{k=1}^{\bar{k}} \gamma^k p_k(s) + \frac{\gamma^{\bar{k}}}{1-\gamma}.$$

The proof is trivial and is based on Lemma C.13 and the following inequality:

$$u(s) = \mathbf{1}_s^\mathsf{T} \left(\mathbf{I} - \gamma P\right)^{-1} \mathbf{1}_s = \sum_{k=0}^{\infty} \mathbf{1}_s^\mathsf{T} (\gamma P)^k \mathbf{1}_s \le \sum_{k=0}^{\infty} \sum_{k=0}^{\infty} p_k(s)$$

4.7 Empirical Evaluation

We experimentally evaluate the proposed approaches on three problems of increasing complexity. We first demonstrate the soundness of the main ideas on the simple chain problem described in the introduction. Then we evaluate the approach on a modified version of the mountain-car problem, a standard benchmark in reinforcement learning. Finally, we describe an application of the method to a blood-management problem, a complex multi-attribute optimization problem.

The empirical results for the chain problem are depicted in Figure 4.6. Here v_{br} is the value function that minimizes the Bellman residual, v_{ls} is a solution of LSPI (Lagoudakis & Parr, 2003). The value function v_{alp} is the solution of (ALP), v_{alp2} is the solution of (ALP–e) expanded by 2 steps, and v_{ralp} is the solution of (ALP-rx). The values d in the relaxed ALP are 1 except the last state in which it is 10. These values are upper bounds, since all states except the last one are transient. These results show that at least in the simple problem, constraint roll-outs and relaxed formulation perform very well.

In the mountain car benchmark (Sutton & Barto, 1998), an underpowered car needs to climb a hill. To do so, it first needs to back up to an opposite hill to gain sufficient momentum. In our modified version of the mountain car, the reward of 1 is received just before the top of the hill. In the traditional formulation, the task stops as soon as the reward is received.

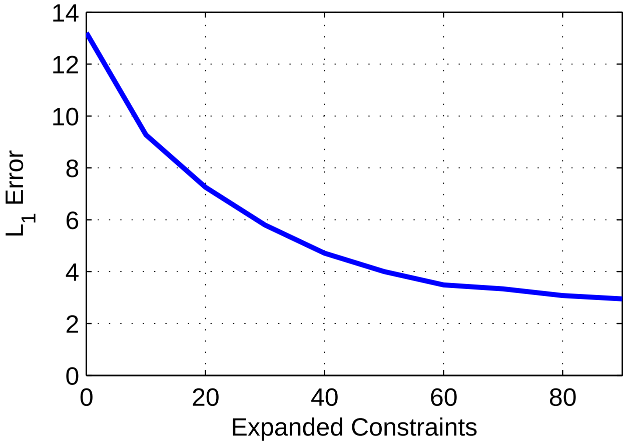

Figure 4.7. Bound on L_1 approximation error with regard to the number of expanded constraints on the mountain car problem.

In our modified problem, the car continues for an extra 0.1 distance, but cannot receive the reward again. The task is described in more detail in Section B.1.3.

We used piecewise linear value function approximation, described in Section 10.2. This representation is in particular suitable for use with ALP, because, unlike tiling, is does not have continuous regions with a constant value. We used 200 randomly chosen states with all actions included per each state. We used 3000 uniformly generated samples. The approximation error of the solution with regard to number of constraints expanded by algorithm 4.1 in Figure 4.7. We evaluated the relaxed linear program formulation on this problem with $d = 0.6 \cdot \mathbf{1}$, which is an upper bound on the dual value, assuming that all states are transient. The average error over 10 runs was 4.474, with only 0.35% of constraints violated. The average discounted return of the ALP policy was 0.338, the average discounted return of policy after 90 expanded constraints was 0.438, and the average discounted return of the relaxed ALP formulation was 0.42.

The blood inventory management problem, described for example in more detail in Section B.2 concerns managing a blood supply inventory and determining the optimal blood-type substitution. A simple greedy solution in this problem achieves return of 70331, standard ALP solution achieves 19108, and the bound on the optimal solution is 94169. Relaxed ALP solution, with the assumption that a loop on any state is at least 20 states long, improves on the greedy solution with value 87055. This is a complex optimization problem and its solution requires that a number of trade-offs, which significantly influence the solution quality.

4.8 Discussion

The modifications that we propose, while improving the performance, increase the computational complexity and rely on prior information. In addition, approximate linear programming is sensitive to the choice of the objective function c and there is very little practical guidance in setting its value. Finally, approximate linear programming minimizes the weighted L_1 norm instead of L_∞ norm required by the online bounds (2.1) and (2.2).

While ALP does not provide as strong bounds on the policy loss as approximate bilinear programming, it is nevertheless worth studying. Approximate linear program is a convex mathematical program, which can be solved in polynomial time and is straightforward to analyze. Finally, when the value function is used with heuristic search, there is no justification for minimizing the Bellman residual; and therefore ALP may be preferable to approximate bilinear programs. We take advantage of this simplicity to derive homotopy continuation methods in Chapter 6, which are used for feature selection in Chapter 10.

4.9 Contributions

The chapter presents a new derivation of approximate linear programs based on minimization of the online error bounds. This formulation generalizes existing work on approximate linear programming — de Farias (2002) in particular — and shows new performance bounds. These bounds can be used to improve the performance of ALP. Theorem 4.2 shows a new performance bound that mildly generalizes existing bounds to a new tighter representation. Theorems 4.5 and 4.4 are new comprehensive approximation error bounds for ALP that include all the representation, sampling and algorithmic components. The results in Sections 4.5 and 4.6 present new improvements over existing methods.

CHAPTER 5

APPROXIMATE BILINEAR PROGRAMMING: TIGHT APPROXIMATION

This chapter shows how to formulate value function approximation as a separable bilinear program. The bilinear program formulation is loosely based on approximate linear programs, but provides a much tighter approximation. We propose and study several closely-related formulations. These formulations minimize the bounds in Theorems 2.16 and 2.17.

The remainder of the chapter is organized as follows. In Section 5.1 we describe the proposed approximate bilinear programming (ABP) formulations and the guarantees it provides. Section 5.2 extends the formulation and guarantees to sampled ABPs. Basic methods for solving bilinear programs are described in Section 5.3, but a more detailed description can be found in Chapter 7.

A drawback of approximate bilinear program formulation is its computational complexity, which may be exponential. We show in Section 5.3 that this is unavoidable, because minimizing the approximation error bound is in fact NP hard. Section 5.4 shows that ABP can be seen as an improvement of approximate linear programming and approximate policy iteration. Section 5.5 experimentally evaluates properties of ABPs on simple benchmark problems.

5.1 Bilinear Program Formulations

This section shows how to formulate value function approximation as a separable bilinear program. Bilinear programs are a generalization of linear programs with an additional bilinear term in the objective function. A separable bilinear program consists of two linear programs with independent constraints and are fairly easy to solve and analyze.

Definition 5.1 (Separable Bilinear Program)**.** A *separable* bilinear program in the normal form is defined as follows:

$$
\begin{array}{llll}
\min\limits_{w,x \,|\, y,z} & s_1^\top w + r_1^\top x + x^\top C y + r_2^\top y + s_2^\top z & & \\
\text{s.t.} & A_1 x + B_1 w = b_1 & A_2 y + B_2 z = b_2 & \text{(BP–m)} \\
 & w, x \geq \mathbf{0} & y, z \geq \mathbf{0} &
\end{array}
$$

The objective of the bilinear program (BP–m) is denoted as $f(w, x, y, z)$. We separate the variables using a vertical line and the constraints using different columns to emphasize the separable nature of the bilinear program. In this thesis, we only use *separable* bilinear programs and refer to them simply as bilinear programs.

We present three different approximate bilinear formulations that minimize the following bounds on the approximate value function.

1. *Robust policy loss*: Minimizes $\|v^* - v_\pi\|_\infty$ by minimizing the bounds in Theorem 2.16:
$$\min_{\pi \in \Pi} \|v^* - v_\pi\|_\infty \leq \min_{v \in \mathcal{M}} \frac{1}{1-\gamma}\|v - Lv\|_\infty$$
2. *Expected policy loss*: Minimizes $\|v^* - v_\pi\|_{1,\alpha}$ by minimizing the bounds in Theorem 2.17:
$$\min_{\pi \in \Pi} \|v^* - v_\pi\|_{1,\alpha} \leq \alpha^{\mathsf{T}} v^* + \min_{v \in \mathcal{M}} \left(-\alpha^{\mathsf{T}} \tilde{v} + \frac{1}{1-\gamma}\|v - Lv\|_\infty \right)$$
$$\min_{\pi \in \Pi} \|v^* - v_\pi\|_{1,\alpha} \leq \alpha^{\mathsf{T}} v^* + \min_{v \in \mathcal{M}} \left(-\alpha^{\mathsf{T}} \tilde{v} + \|v - Lv\|_{1,\bar{u}(v)} \right).$$
3. *The sum of k largest errors*: This formulation represents a hybrid between the robust and expected formulations. It is more robust than simply minimizing the expected performance but is not as sensitive to worst-case performance.

The appropriateness of each formulation depends on the particular circumstances of the domain. For example, minimizing robust bounds is advantageous when the initial distribution is not known and the performance must be consistent under all circumstances. On the other hand, minimizing expected bounds on the value function is useful when the initial distribution is known.

The expected policy loss minimization is related to the relaxed formulation of the approximate linear program Equation ALP-r. As we show below, the bilinear program formulation is tighter and does not require as much prior knowledge. In fact, we also propose a formulation that does not require the bound on the state–visitation frequencies $\bar{u}$ of the greedy policy.

In the formulations described below, we initially assume that samples of all states and actions are used. This means that the precise version of the operator L is available. To solve large problems, the number of samples would be much smaller; either simply subsampled or reduced using the structure of the MDP. Reducing the number of constraints in linear programs corresponds to simply removing constraints. In approximate bilinear programs it also reduces the number of some variables, as Section 5.2 describes.

The formulations below denote the value function approximation generically by $v \in \mathcal{M}$. That restricts the value functions to be representable using features. Representable value functions v can be replaced by a set of variables x as $v = \Phi x$. This reduces the number of variables to the number of features.

5.1.1 Robust Policy Loss

The solution of the robust approximate bilinear program minimizes the L_∞ norm of the Bellman residual $\|v - Lv\|_\infty$. This minimization can be formulated as follows.

$$\begin{array}{llll}
\min\limits_{\pi \,|\, \lambda,\lambda',v} & \pi^{\mathsf{T}}\lambda + \lambda' & & \\
\text{s.t.} & B\pi = \mathbf{1} & Av - b \geq \mathbf{0} & \\
 & \pi \geq \mathbf{0} & \lambda + \lambda'\mathbf{1} \geq Av - b & \qquad \text{(ABP–}L_\infty\text{)} \\
 & & \lambda, \lambda' \geq \mathbf{0} & \\
 & & v \in \mathcal{M} &
\end{array}$$

All the variables are vectors except λ', which is a scalar. The matrix A represents constraints that are identical to the constraints in (ALP). The variables λ correspond to all state-action pairs. These variables represent the Bellman residuals that are being minimized. This formulation offers the following guarantees.

Theorem 5.2. *Given Assumption 2.21, any optimal solution* $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ *of the approximate bilinear program* (ABP–L_∞) *satisfies:*

$$\begin{aligned}\tilde{\pi}^\mathsf{T}\tilde{\lambda} + \tilde{\lambda}' = \|L\tilde{v} - \tilde{v}\|_\infty &\le \min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty \\ &\le 2 \min_{v \in \mathcal{M}} \|Lv - v\|_\infty \\ &\le 2(1+\gamma) \min_{v \in \mathcal{M}} \|v - v^*\|_\infty .\end{aligned}$$

Moreover, there exists an optimal solution $\tilde{\pi}$ *that is greedy with respect to* $\tilde{v}$ *for which the policy loss is bounded by:*

$$\|v^* - v_{\tilde{\pi}}\|_\infty \le \frac{2}{1-\gamma} \left(\min_{v \in \mathcal{M}} \|Lv - v\|_\infty \right).$$

The proof of the theorem can be found in Section C.6. It is important to note that the theorem states that solving the approximate bilinear program is equivalent to minimization over *all* representable value functions, not only the transitive-feasible ones. This follows by subtracting a constant vector $\mathbf{1}$ from $\tilde{v}$ to balance the lower bounds on the Bellman residual error with the upper ones. This reduces the Bellman residual by $1/2$ without affecting the policy. Finally, note that whenever $v^* \in \mathcal{M}$, both ABP and ALP will return the optimal value function v^*. The theorem follows from the following lemmas.

First, define the following linear program, which solves for the L_∞ norm of the Bellman update L_π for value function v and policy π.

$$\begin{aligned} f_1(\pi, v) = \min_{\lambda, \lambda'} \quad & \pi^\mathsf{T}\lambda + \lambda' \\ \text{s.t.} \quad & \mathbf{1}\lambda' + \lambda \ge Av - b \\ & \lambda \ge \mathbf{0} \end{aligned} \tag{5.1}$$

The linear program (5.1) corresponds to the bilinear program (ABP–L_∞) with a fixed policy π and value function v.

Lemma 5.3. *Let* $v \in \mathcal{K}$ *be a transitive-feasible value function and let* π *be a policy. Then:*

$$f_1(\pi, v) \ge \|v - L_\pi v\|_\infty,$$

with an equality for a deterministic *policy* π*.*

The proof of the lemma can be found in Section C.6.

When the policy π is fixed, the approximate bilinear program (ABP–L_∞) becomes the following linear program:

$$\begin{aligned} f_2(\pi) = \min_{\lambda,\lambda',v} \quad & \pi^{\mathsf{T}}\lambda + \lambda' \\ \text{s.t.} \quad & Av - b \geq \mathbf{0} \\ & \mathbf{1}\lambda + \lambda' \geq Av - b \\ & \lambda \geq \mathbf{0} \\ & v \in \mathcal{M} \end{aligned} \tag{5.2}$$

Using Lemma C.19 , this linear program corresponds to:

$$f_2(\pi) = \min_{v\in\mathcal{M}\cap\mathcal{K}} f_1(\pi, v).$$

Then it is easy to show that:

Lemma 5.4. *Given a policy π, let $\tilde{v}$ be an optimal solution of the linear program* (5.2)*. Then:*

$$f_2(\pi) = \|L_\pi\tilde{v} - \tilde{v}\|_\infty \leq \min_{v\in\mathcal{M}\cap\mathcal{K}} \|L_\pi v - v\|_\infty.$$

When v is fixed, the approximate bilinear program (ABP–L_∞) becomes the following linear program:

$$\begin{aligned} f_3(v) = \min_{\pi} \quad & f_2(\pi, v) \\ \text{s.t.} \quad & B\pi = \mathbf{1} \\ & \pi \geq \mathbf{0} \end{aligned} \tag{5.3}$$

Note that the program is only meaningful if v is transitive-feasible and that the function f_2 corresponds to a minimization problem.

Lemma 5.5. *Let $v \in \mathcal{M}\cap\mathcal{K}$ be a transitive-feasible value function. There exists an optimal solution $\tilde{\pi}$ of the linear program* (5.3) *such that:*

1. *$\tilde{\pi}$ represents a* deterministic *policy*
2. *$L_{\tilde{\pi}}v = Lv$*
3. *$\|L_{\tilde{\pi}}v - v\|_\infty = \min_{\pi\in\Pi}\|L_\pi v - v\|_\infty = \|Lv - v\|_\infty$*

The proof of the lemma can be found in Section C.6.

5.1.2 Expected Policy Loss

This section describes bilinear programs that minimize expected policy loss for a given initial distribution $\|v - Lv\|_{1,\alpha}$. The initial distribution can be used to derive tighter bounds on the policy loss. We describe two formulations. They respectively minimize an L_∞ and a weighted L_1 norm on the Bellman residual.

The expected policy loss can be minimized by solving the following bilinear formulation.

$$
\begin{array}{llll}
\min\limits_{\pi \,|\, \lambda,\lambda',v} & \pi^\mathsf{T}\lambda + \lambda' - (1-\gamma)\alpha^\mathsf{T} v & & \\
\text{s.t.} & B\pi = \mathbf{1} & Av - b \geq \mathbf{0} & \\
& \pi \geq \mathbf{0} & \lambda + \lambda'\mathbf{1} \geq Av - b & \qquad\text{(ABP–}L_1\text{)} \\
& & \lambda, \lambda' \geq \mathbf{0} & \\
& & v \in \mathcal{M} &
\end{array}
$$

Notice that this formulation is identical to the bilinear program (ABP–L_∞) with the exception of the term $-(1-\gamma)\alpha^\mathsf{T} v$.

Theorem 5.6. *Given Assumption 2.21, any optimal solution $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ of the approximate bilinear program* (ABP–L_1) *satisfies:*

$$
\begin{aligned}
\frac{1}{1-\gamma}\left(\tilde{\pi}^\mathsf{T}\tilde{\lambda} + \tilde{\lambda}'\right) - \alpha^\mathsf{T}\tilde{v} = \frac{1}{1-\gamma}\|L\tilde{v} - \tilde{v}\|_\infty - \alpha^\mathsf{T} v &\leq \min_{v \in \mathcal{K} \cap \mathcal{M}} \left(\frac{1}{1-\gamma}\|Lv - v\|_\infty - \alpha^\mathsf{T} v\right) \\
&\leq \min_{v \in \mathcal{M}} \left(\frac{1}{1-\gamma}\|Lv - v\|_\infty - \alpha^\mathsf{T} v\right)
\end{aligned}
$$

Moreover, there exists an optimal solution $\tilde{\pi}$ that is greedy with respect to $\tilde{v}$ for which the policy loss is bounded by:

$$
\|v^* - v_{\tilde{\pi}}\|_{1,\alpha} \leq \frac{2}{1-\gamma}\left(\min_{v \in \mathcal{M}} \frac{1}{1-\gamma}\|Lv - v\|_\infty - \|v^* - v\|_{1,\alpha}\right).
$$

The proof of the theorem can be found in Section C.6. Notice that the bound in this theorem is tighter than the one in Theorem 5.2. In particular, $\|v^* - \tilde{v}\|_{1,\alpha} > 0$, unless the solution of the ABP is the optimal value function.

The bilinear program formulation in (ABP–L_1) can be further strengthened when an upper bound on the state-visitation frequencies is available.

$$
\begin{array}{llll}
\min\limits_{\pi \,|\, \lambda,v} & \pi^\mathsf{T} U\lambda - \alpha^\mathsf{T} v & & \\
\text{s.t.} & B\pi = \mathbf{1} & Av - b \geq \mathbf{0} & \\
& \pi \geq \mathbf{0} & \lambda = Av - b & \qquad\text{(ABP–}U\text{)} \\
& & v \in \mathcal{M} &
\end{array}
$$

Here $U : |\mathcal{S}| \cdot |\mathcal{A}| \times |\mathcal{S}| \cdot |\mathcal{A}|$ is a matrix that maps a policy to bounds on state-action visitation frequencies. It must satisfy that:

$$
\pi(s,a) = 0 \Rightarrow (\pi^\mathsf{T} U)(s,a) = 0 \quad \forall s \in \mathcal{S}\ \forall a \in \mathcal{A}.
$$

Remark 5.7. One simple option is to have U represent a diagonal matrix of $\bar{u}$, where $\bar{u}$ is the bound for all policies $\pi \in \Pi$. That is:

$$
U((s,a),(s',a')) = \begin{cases} \bar{u}(s) & s' = s \\ 0 & \text{otherwise} \end{cases} \qquad \forall s, s' \in \mathcal{S}\ a, a' \in \mathcal{A}.
$$

The formal guarantees for this formulation are as follows.

Theorem 5.8. *Given Assumption 2.21 and that for all $\pi \in \Pi \,:\, \sum_{a\in\mathcal{A}}(\pi^\mathsf{T} U)(s,a) \geq u_\pi^\mathsf{T}(s)$, any optimal solution $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda'})$ of the bilinear program (ABP–U) then satisfies:*

$$\tilde{\pi}^\mathsf{T} U \tilde{\lambda} - \alpha^\mathsf{T} \tilde{v} = \|\tilde{v} - L\tilde{v}\|_{\bar{u}} - \alpha^\mathsf{T}\tilde{v} \leq \min_{v\in\mathcal{K}\cap\mathcal{M}} \left(\|v - Lv\|_{\bar{u}} - \alpha^\mathsf{T} v \right).$$

Assuming that U is defined as in Remark 5.7, there exists an optimal solution $\tilde{\pi}$ that is greedy with respect to $\tilde{v}$ and:

$$\begin{aligned} \|v^* - v_{\tilde{\pi}}\|_{1,\alpha} &\leq \min_{v\in\mathcal{M}\cap\mathcal{K}} \left(\|v - Lv\|_{1,\bar{u}(v)} - \|v^* - v\|_{1,\alpha} \right) \\ &\leq \max\{1, \gamma \bar{u}(v)^\mathsf{T}\mathbf{1}\} \min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - v^*\|_\infty \\ &\leq \left(2 + \gamma + \mathbf{1}^\mathsf{T}\bar{u}(v) - \frac{1}{1-\gamma} \right) \min_{v\in\mathcal{M}} \|v - v^*\|_\infty \end{aligned}$$

Here, $\bar{u}(v)$ represents an upper bound on the state-action visitation frequencies for a policy greedy with respect to value function v.

Unlike Theorems 5.2 and 5.6, the bounds in this theorem do not guarantee that the solution quality does not degrade by restricting the value function to be transitive-feasible. Also,

To prove the theorem we first define the following linear program that solves for the L_1 norm of the Bellman update L_π for value function v.

$$\begin{aligned} f_1(\pi, v) = \min_{\lambda,\lambda'} \quad & \pi^\mathsf{T} U \lambda \\ \text{s.t.} \quad & \mathbf{1}\lambda' + \lambda \geq Av - b \\ & \lambda \geq \mathbf{0} \end{aligned} \tag{5.4}$$

The linear program (5.1) corresponds to the bilinear program (ABP–U) with a fixed policy π and value function v. Notice, in particular, that $\alpha^\mathsf{T} v$ is a constant.

Lemma 5.9. *Let value function v be feasible in the bilinear program (ABP–U), and let π be an arbitrary policy. Then:*

$$f_1(\pi, v) \geq \|L_\pi v - v\|_{1,\bar{u}},$$

with an equality for a deterministic *policy.*

The proof of the lemma can be found in Section C.6.

The proof of Theorem 5.8 is similar to the proof of Theorem 5.2, but uses Theorem 2.17 instead of Theorem 2.16 to bound the policy loss and relies on Lemma 5.9. The existence of a deterministic and greedy optimal solution $\tilde{\pi}$ follows also like Theorem 5.2, omitting λ' and weighing λ by $\bar{u}$. The last two inequalities follow from the following lemma.

Lemma 5.10. *The following inequalities hold for any representable and transitive–feasible value functions:*

$$\begin{aligned} \min_{v\in\mathcal{M}\cap\mathcal{K}} \left(\|v - Lv\|_{1,\bar{u}(v)} - \|v^* - v\|_{1,\alpha} \right) &\leq \max\{1, \gamma\bar{u}(v)^\mathsf{T}\mathbf{1}\} \min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - v^*\|_\infty \\ &\leq \left(2 + \gamma + \mathbf{1}^\mathsf{T}\bar{u}(v) - \frac{1}{1-\gamma} \right) \min_{v\in\mathcal{M}} \|v - v^*\|_\infty \end{aligned}$$

The proof of the lemma can be found in Section C.6.

5.1.3 Hybrid Formulation

While the robust bilinear formulation (ABP–L_∞) guarantees to minimize the robust error it may be overly pessimistic. The bilinear program (ABP–U), on the other hand, optimizes the average performance, but does not provide strong guarantees. It is possible to combine the advantages (and disadvantaged) of these programs using a hybrid formulation. The hybrid formulation minimizes the average over the largest elements with weight at most k of the Bellman residual. To do that, define:

$$\|x\|_{k,c} = \max_{\{y \,|\, \mathbf{1}^\mathsf{T} y = k, \mathbf{1} \geq y \geq \mathbf{0}\}} \sum_{i=1}^{n} y(i)c(i)|x(i)|,$$

where n is the length of vector x and $c \geq \mathbf{0}$. It is easy to show that this norm represents the c-weighted L_1 norm of the k largest components of the vector. As such, it is more robust than the plain L_1 norm, but is not as sensitive to outliers as the L_∞ norm. Notice that the solution may be fractional when $k \notin \mathbb{Z}$ — that is, some elements are counted only partially.

The bilinear program that minimizes the hybrid norm is defined as follows.

$$\begin{array}{lll}
\min\limits_{\pi \,|\, \lambda, \lambda', v} & \pi^\mathsf{T} U \lambda + k\lambda' & \\
\text{s.t.} & B\pi = \mathbf{1} & Av - b \geq \mathbf{0} \\
 & \pi \geq \mathbf{0} & \lambda + \lambda' U^{-1} \mathbf{1} \geq Av - b \\
 & & \lambda, \lambda' \geq \mathbf{0} \\
 & & v \in \mathcal{M}
\end{array} \qquad \text{(ABP–h)}$$

Here U is a matrix that maps a policy to bounds on state-action visitation frequencies, for example, as defined in Remark 5.7.

Theorem 5.11. *Given Assumption 2.21 and U that is defined as in Remark 5.7, any optimal solution $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ of (ABP–h) then satisfies:*

$$\tilde{\pi}^\mathsf{T} U \tilde{\lambda} + k\tilde{\lambda}' = \|L\tilde{v} - \tilde{v}\|_{k,\bar{u}(\tilde{\pi})} \leq \min_{v \in \mathcal{M} \cap \mathcal{K}} \|L\tilde{v} - \tilde{v}\|_{k,\bar{u}(v)}.$$

Here, $\bar{u}(v)$ represents the upper bound on the state-action visitation frequencies for policy greedy with respect to value function v.

The implication of these bounds on the policy loss are not yet known, but it is likely that some form of policy loss bounds can be developed.

The proof of the theorem is almost identical to the proof of Theorem 5.2 lemma. We first define the following linear program, which solves for the required norm of the Bellman update L_π for value function v and policy π.

$$\begin{array}{ll}
f_1(\pi, v) = \min\limits_{\lambda, \lambda'} & \pi^\mathsf{T} U \lambda + k\lambda' \\
\text{s.t.} & \lambda' U^{-1} \mathbf{1} + \lambda \geq Av - b \\
 & \lambda, \lambda' \geq \mathbf{0}
\end{array} \qquad (5.5)$$

The linear program (5.5) corresponds to the bilinear program (ABP–h) with a fixed policy π and value function v.

Lemma 5.12. *Let $v \in \mathcal{K}$ be a transitive-feasible value function and let π be a policy and U be defined as in Remark 5.7. Then:*

$$f_1(\pi, v) \geq \|v - L_\pi v\|_{k,\bar{u}},$$

with an equality for a deterministic *policy π.*

The proof of the lemma can be found in Section C.6.

5.2 Sampling Guarantees

In most practical problems, the number of states is too large to be explicitly enumerated. Even though the value function is restricted to be representable, the problem cannot be solved. The usual approach is to sample a limited number of states, actions, and their transitions to approximately calculate the value function. This section shows basic properties of the samples that can provide guarantees of the solution quality with incomplete samples.

Theorem 5.2 shows that sampled robust ABP minimizes $\|v-\tilde{L}v\|_\infty$ or $\|v-\bar{L}v\|_\infty$, depending on the samples used. It is then easy to derive sampling bounds that rely on the sampling assumptions defined above.

Theorem 5.13. *Let the optimal solutions to the sampled and precise Bellman residual minimization problems be:*

$$v_1 \in \arg\min_{v \in \mathcal{M} \cap \mathcal{K}} \|v - Lv\|_\infty \qquad v_2 \in \arg\min_{v \in \mathcal{M} \cap \mathcal{K}} \|v - \hat{L}v\|_\infty \qquad v_3 \in \arg\min_{v \in \mathcal{M} \cap \mathcal{K}} \|v - \tilde{L}v\|_\infty$$

Value functions v_1, v_2, v_3 correspond to solutions of instances of robust approximate bilinear programs for the given samples. Also let $\hat{v}_i = v_{\pi_i}$, where π_i is greedy with respect to v_i. Then, given Assumptions 2.21, 2.26, and 2.28, the following holds:

$$\begin{aligned}
\|v^* - \hat{v}_1\|_\infty &\leq \frac{2}{1-\gamma} \min_{v \in \mathcal{M}} \|v - Lv\|_\infty \\
\|v^* - \hat{v}_2\|_\infty &\leq \frac{2}{1-\gamma} \left(\min_{v \in \mathcal{M}} \|v - Lv\|_\infty + \epsilon_p \right) \\
\|v^* - \hat{v}_3\|_\infty &\leq \frac{2}{1-\gamma} \left(\min_{v \in \mathcal{M}} \|v - Lv\|_\infty + \epsilon_p + 2\epsilon_s \right)
\end{aligned}$$

The proof of the theorem can be found in Section C.6.

These bounds show that it is possible to bound policy loss due to incomplete samples. As mentioned above, existing bounds on constraint violation in approximate linear programming (de Farias & van Roy, 2004) typically do not easily lead to policy loss bounds. The bounds are also much tighter than Theorem 4.3, since the ϵ terms do not involve $1/(1-\gamma)$, which can be very large.

Sampling guarantees for other bilinear program formulations are very similar. Because they also rely on an approximation of the initial distribution and the policy loss, they require additional assumptions on uniformity of state-samples.

Algorithm 5.1: Iterative algorithm for solving (BP–m)

```
$(x_0, w_0) \leftarrow$ random ;
$(y_0, z_0) \leftarrow \arg\min_{y,z} f(w_0, x_0, y, z)$ ;
$i \leftarrow 1$ ;
while $y_{i-1} \neq y_i$ or $x_{i-1} \neq x_i$ do
    $(y_i, z_i) \leftarrow \arg\min_{\{y,z \,|\, A_2 y + B_2 z = b_2 \; y,z \geq \mathbf{0}\}} f(w_{i-1}, x_{i-1}, y, z)$ ;
    $(x_i, w_i) \leftarrow \arg\min_{\{x,w \,|\, A_1 x + B_1 w = b_1 \; x,w \geq \mathbf{0}\}} f(w, x, y_i, z_i)$ ;
    $i \leftarrow i + 1$
return $f(w_i, x_i, y_i, z_i)$
```

To summarize, this section identifies basic assumptions on the sampling behavior and shows that approximate bilinear programming scales well in the face of uncertainty caused by incomplete sampling. More detailed analysis in Chapter 9 focuses on identifying problem-specific assumptions and sampling modes that guarantee the basic conditions, namely satisfying Assumptions 2.28 and 2.26.

5.3 Solving Bilinear Programs

This section describes methods for solving approximate bilinear programs. Bilinear programs can be easily mapped to other global optimization problems, such as mixed integer linear programs (Horst & Tuy, 1996). We focus on a simple iterative algorithm for solving bilinear programs approximately, which also serves as a basis for many optimal algorithms.

Solving a bilinear program is an NP-complete problem (Bennett & Mangasarian, 1992). The membership in NP follows from the finite number of basic feasible solutions of the individual linear programs, each of which can be checked in polynomial time. The NP-hardness is shown by a reduction from the SAT problem.

There are two main approaches to solving bilinear programs optimally. In the first approach, a relaxation of the bilinear program is solved. The solution of the relaxed problem represents a lower bound on the optimal solution. The relaxation is then iteratively refined, for example by adding cutting plane constraints, until the solution becomes feasible. This is a common method used to solve integer linear programs. The relaxation of the bilinear program is typically either a linear or semi-definite program (Carpara & Monaci, 2009).

In the second approach, feasible, but suboptimal, solutions of the bilinear program are calculated approximately. The approximate algorithms are usually some variation of algorithm 5.1. The bilinear program formulation is then refined — using concavity cuts (Horst & Tuy, 1996) — to eliminate previously computed feasible solutions and solved again. This procedure can be shown to find the optimal solution by eliminating all suboptimal feasible solutions.

The most common and simplest approximate algorithm for solving bilinear programs is algorithm 5.1. This algorithm is shown for the general bilinear program (BP–m), where $f(w, x, y, z)$ represents the objective function. The minimizations in the algorithm are linear programs which can be easily solved. Interestingly, as we will show in Section 5.4, algorithm 5.1 applied to ABP generalizes a version of API.

While algorithm 5.1 is not guaranteed to find an optimal solution, its empirical performance is often remarkably good (Mangasarian, 1995). Its basic properties are summarized by the following proposition.

Proposition 5.14 (e.g. (Bennett & Mangasarian, 1992))**.** *algorithm 5.1 is guaranteed to converge, assuming that the linear program solutions are in a vertex of the optimality simplex. In addition, the global optimum is a fixed point of the algorithm, and the objective value monotonically improves during execution.*

The proof is based on the finite count of the basic feasible solutions of the individual linear programs. Because the objective function does not increase in any iteration, the algorithm will eventually converge.

algorithm 5.1 can be further refined in case of approximate bilinear programs. For example, the constraint $v \in \mathcal{M}$ in the bilinear programs serves just to simplify the bilinear program and a value function that violates it may still be acceptable. The following proposition motivates the construction of a new value function from two transitive-feasible value functions.

Proposition 5.15. *Let $\tilde{v}_1$ and $\tilde{v}_2$ be feasible value functions in* (ABP–L_∞)*. Then the value function*

$$\tilde{v}(s) = \min\{\tilde{v}_1(s), \tilde{v}_2(s)\}$$

is also feasible in bilinear program (ABP–L_∞)*. Therefore $\tilde{v} \geq v^*$ and*

$$\|v^* - \tilde{v}\|_\infty \leq \min\left\{\|v^* - \tilde{v}_1\|_\infty, \|v^* - \tilde{v}_2\|_\infty\right\}.$$

The proof of the proposition can be found in Section C.6. The proof of the proposition is based on Jensen's inequality and is provided in the appendix. Note that $\tilde{v}$ may have a greater Bellman residual than either $\tilde{v}_1$ or $\tilde{v}_2$.

Proposition 5.15 can be used to extend algorithm 5.1 when solving ABPs. One option is to take the state-wise minimum of values from multiple random executions of algorithm 5.1, which preserves the transitive feasibility of the value function. However, the increasing number of value functions used to obtain $\tilde{v}$ also increases the potential sampling error.

5.4 Discussion and Related ADP Methods

This section describes connections between approximate bilinear programming and two closely related approximate dynamic programming methods: ALP, and L_∞-API, which are commonly used to solve factored MDPs (Guestrin et al., 2003). Our analysis sheds light on some of their observed properties and leads to a new *convergent* form of approximate policy iteration.

Approximate bilinear programming addresses some important issues with ALP: 1) ALP provides value function bounds with respect to L_1 norm, which does not guarantee small policy loss, 2) ALP's solution quality depends significantly on the heuristically-chosen objective function c in (ALP) (de Farias, 2002), 3) the performance bounds involve a constant $1/(1-\gamma)$ which can be very large when γ is close to 1 and 4) incomplete constraint samples in ALP easily lead to unbounded linear programs. The drawback of using approximate bilinear programming, however, is the higher computational complexity.

The first and the second issue in ALP can be addressed by choosing a problem-specific objective function c (de Farias, 2002). Unfortunately, all existing bounds require that c is chosen based on the optimal ALP solution for c. This is impossible to compute in practice. Heuristic values for c are used instead. Robust approximate bilinear program (ABP–L_∞), on the other hand, has no such parameters. On the other hand, the expected-loss bilinear program (ABP–U) can be seen as a method for simultaneously optimizing c and the approximate linear program.

The fourth issue in approximate linear programs arises when the constraints need to be sampled. The ALP may become unbounded with incomplete samples because its objective value is defined using the L_1 norm on the value function, and the constraints are defined using the L_∞ norm of the Bellman residual. In approximate bilinear programs, the Bellman residual is used in both the constraints and objective function. The objective function of ABP is then bounded below by 0 for an arbitrarily small number of samples.

The NP-completeness of ABP compares unfavorably with the polynomial complexity of ALP. However, most other approximate dynamic programming algorithms are not guaranteed to converge to a solution in finite time. As we show below, the exponential time complexity of ABP is unavoidable (unless P = NP).

The following theorem shows that the computational complexity of the ABP formulation is asymptotically the same as the complexity of tightly approximating the value function.

Theorem 5.16. *Assume* $0 < \gamma < 1$*, and a given* $\epsilon > 0$*. Then it is NP-complete to determine:*

$$\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty < \epsilon \qquad \min_{v \in \mathcal{M}} \|Lv - v\|_\infty < \epsilon.$$

The problem remains NP-complete when Assumption 2.21 is satisfied. It is also NP-complete to determine:

$$\min_{v \in \mathcal{M}} \|Lv - v\|_\infty - \|v^* - v\|_{1,\alpha} < \epsilon \qquad \min_{v \in \mathcal{M}} \|Lv - v\|_{1,\bar{u}} - \|v^* - v\|_{1,\alpha} < \epsilon,$$

assuming that $\bar{u}$ *is defined as in Remark 5.7.*

The proof of the theorem can be found in Section C.6.1. As the theorem states, the value function approximation does not become computationally simpler even when Assumption 2.21 holds. Notice that ALP can determine whether $\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty = 0$ in polynomial time.

The proof of Theorem 5.16 is based on a reduction from SAT and can be found in Section C.6.1. The policy in the reduction determines the true literal in each clause, and the approximate value function corresponds to the truth value of the literals. The approximation basis forces literals that share the same variable to have consistent values.

Approximate bilinear programming can also improve on API with L_∞ minimization (L_∞-API for short), which is a leading method for solving factored MDPs (Guestrin et al., 2003). Minimizing the L_∞ approximation error is theoretically preferable, since it is compatible with the existing bounds on policy loss (Guestrin et al., 2003). The bounds on value function approximation in API are typically (Munos, 2003):

$$\limsup_{k\to\infty} \|v^* - \hat{v}_k\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{k\to\infty} \|\tilde{v}_k - v_k\|_\infty.$$

The bounds are described in more detail in . These bounds are looser than the bounds on solutions of ABP by at least a factor of $1/(1-\gamma)$. Often the difference may be up to $1/(1-\gamma)^2$ since the error $\|\tilde{v}_k - v_k\|_\infty$ may be significantly larger than $\|\tilde{v}_k - L\tilde{v}_k\|_\infty$. Finally, the bounds cannot be easily used, because they only hold in the limit.

We propose *Optimistic Approximate Policy Iteration* (OAPI), a modification of API (shown in algorithm 3.3). In OAPI, $\mathcal{Z}(\pi)$ is calculated using the following optimization problem:

$$\begin{aligned} \min_{\phi,v} \quad & \phi \\ \text{s.t.} \quad & Av \geq b \quad (\equiv (\mathbf{I} - \gamma P_\pi)v \geq r_\pi \ \ \forall \pi \in \Pi) \\ & -(\mathbf{I} - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{aligned} \tag{5.6}$$

In fact, OAPI corresponds to algorithm 5.1 applied to ABP because the linear program (5.6) corresponds to (ABP–L_∞) with a fixed π (see (5.2)). Then, using Proposition 5.14, we get the following corollary.

Corollary 5.17. *Optimistic approximate policy iteration converges in finite time. In addition, the Bellman residual of the generated value functions monotonically decreases.*

OAPI differs from L_∞-API in two ways: 1) OAPI constrains the Bellman residuals by 0 from below and by ϕ from above, and then it minimizes ϕ. L_∞-API constrains the Bellman residuals by ϕ from both above and below. 2) OAPI, like API, uses only the current policy for the upper bound on the Bellman residual, but uses *all* the policies for the lower bound on the Bellman residual.

L_∞-API cannot return an approximate value function that has a lower Bellman residual than ABP, given the optimality of ABP described in Theorem 5.2. However, even OAPI, an approximate ABP algorithm, performs comparably to L_∞-API, as the following theorem states.

Theorem 5.18. *Assume that L_∞-API converges to a policy π and a value function v that both satisfy: $\phi = \|v - L_\pi v\|_\infty = \|v - Lv\|_\infty$. Then*

$$\tilde{v} = v + \frac{\phi}{1-\gamma}\mathbf{1}$$

is feasible in the bilinear program (ABP–L_∞)*, and it is a fixed point of OAPI. In addition, the greedy policies with respect to $\tilde{v}$ and v are identical.*

The proof of the theorem is below. Notice that while the optimistic and standard policy iterations can converge to the same solutions, the steps in their computation may not be identical. The actual results will depend on the initialization.

To prove the theorem, we first consider L_∞-API$_2$ as a modification of L_∞-API. L_∞-API$_2$ is shown in algorithm 3.3, where $\mathcal{Z}(\pi)$ is calculated using the following program:

$$\begin{aligned} \min_{\phi,v} \quad & \phi \\ \text{s.t.} \quad & (\mathbf{I} - \gamma P_a)v + \mathbf{1}\phi \geq r_a \quad \forall a \in \mathcal{A} \\ & -(\mathbf{I} - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{aligned} \tag{5.7}$$

The difference between linear programs (3.5) and (5.7) is that (3.5) involves only the current policy, while (5.7) bounds $(\mathbf{I} - \gamma P_a)v + \mathbf{1}\phi \geq r_a$ from below for all policies. Linear program (5.7) differs from linear program (5.6) by not bounding the Bellman residual from below by $\mathbf{0}$.

Proposition 5.19. *L_∞-API and L_∞-API$_2$ generate the same sequence of policies if the initial policies and tie-breaking is the same.*

Proof. The proposition follows simply by induction from Lemma C.20. The basic step follows directly from the assumption. For the inductive step, let $\pi^1_i = \pi^2_i$, where π^1 and π^2 are the policies with (3.5) and (5.7). Then from Lemma C.20, we have that the corresponding value functions $v^1_i = v^2_i + c\mathbf{1}$. Because π^1_{i+1} and π^2_{i+1} are chosen greedily, we have that $\pi^1_{i+1} = \pi^2_{i+1}$. □

The proof of Theorem 5.18 follows.

Proof. The proof is based on two facts. First, $\tilde{v}$ is feasible with respect to the constraints in (ABP–L_∞). The Bellman residual changes for all the policies identically, since a constant vector is added. Second, because L_π is greedy with respect to $\tilde{v}$, we have that $\tilde{v} \geq L_\pi \tilde{v} \geq L\tilde{v}$. The value function $\tilde{v}$ is therefore transitive-feasible.

From Proposition 5.19, L_∞-API can be replaced by L_∞-API$_2$, which will converge to the same policy π. L_∞-API$_2$ will converge to the value function

$$\tilde{v} = v + \frac{\phi}{1-\gamma}\mathbf{1}.$$

From the constraints in (5.7) we have that $\tilde{v} \geq L_\pi \tilde{v}$. Then, since π is the greedy policy with regard to this value function, we have that $\tilde{v} \geq L_\pi \tilde{v} \geq L\tilde{v}$. Thus $\tilde{v}$ is transitive-feasible and feasible in (BP–m) according to Lemma C.19. The theorem then follows from Lemma C.20 and from the fact that the greedy policy minimizes the Bellman residual, as in the proof of item 5.5. □

To summarize, OAPI guarantees convergence, while matching the performance of L_∞-API. The convergence of OAPI is achieved because given a non-negative Bellman residual, the greedy policy also minimizes the Bellman residual. Because OAPI ensures that the Bellman residual is always non-negative, it can progressively reduce it. In comparison, the greedy policy in L_∞-API does not minimize the Bellman residual, and therefore L_∞-API does not always reduce it. Theorem 5.18 also explains why API provides better solutions than ALP, as observed in (Guestrin et al., 2003). From the discussion above, ALP can be seen as an L_1-norm approximation of a single iteration of OAPI. L_∞-API, on the other hand, performs many such ALP-like iterations.

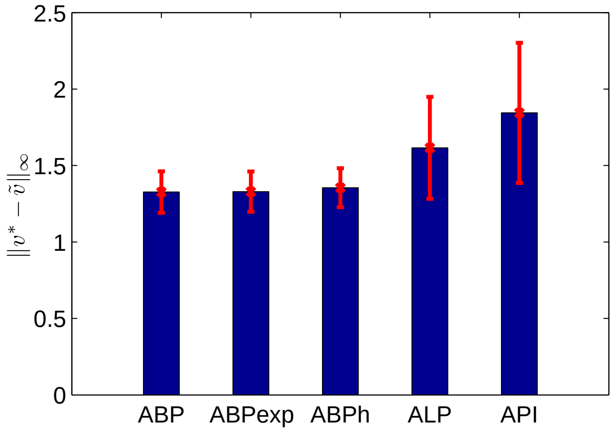

Figure 5.1. L_∞ Bellman residual for the chain problem

5.5 Empirical Evaluation

In this section, we validate the approach by applying it to simple reinforcement learning benchmark problems. The experiments are intentionally designed to avoid interaction between the approximation in the formulation and approximate solution methods. As Theorem 5.18 shows, even OAPI, the very simple approximate algorithm for ABP, can perform as well as existing methods on factored MDPs.

ABP is an off-policy approximation method, like LSPI (Lagoudakis & Parr, 2003) or ALP. That means that the samples can be gathered independently of the control policy. It is necessary, though, that multiple actions are sampled for each state to enable the selection of different policies.

First, we demonstrate and analyze the properties of ABP on a simple chain problem with 200 states, described in Section B.1.2. We experimented with the full state-action sample and randomly chose the features. All results are averages over 50 runs with 15 features. In the results, we use ABP to denote a close-to-optimal solution of robust ABP, ABPexp for the bilinear program (ABP–L_1), and ABPh for (ABP–h) with $k = 5$. API denotes approximate policy iteration that minimizes the L_2 norm.

Figure 5.1 shows the Bellman residual attained by the methods. It clearly shows that the robust bilinear formulation most reliably minimizes the Bellman residual. The other two bilinear formulations are not much worse. Notice also the higher standard deviation of ALP and API. Figure 5.2 shows the expected policy loss, as specified in Definition 2.10, for the calculated value functions. It confirms that the ABP formulation outperforms the robust formulation, since its explicit objective is to minimize the expected loss. Similarly, Figure 5.3 shows the robust policy loss. As expected, it confirms the better performance of the robust ABP formulation in this case.

Note that API and ALP may achieve lower policy loss on this particular domain than ABP formulations, even though their Bellman residual is significantly higher. This is possible since ABP simply minimizes bounds on the policy loss.

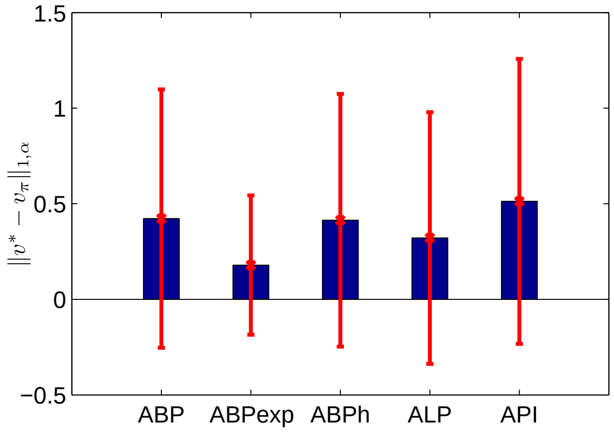

Figure 5.2. Expected policy loss for the chain problem

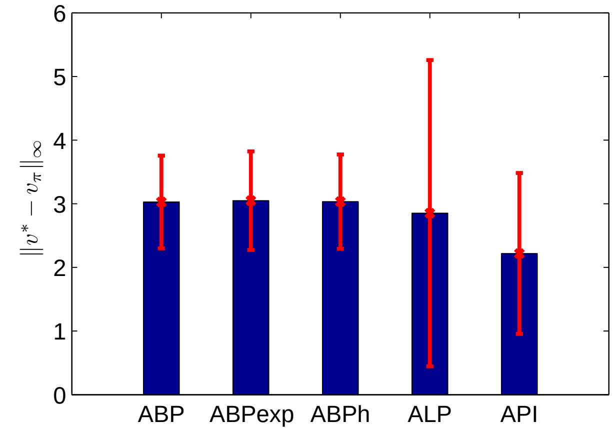

Figure 5.3. Robust policy loss for the chain problem

In the mountain-car benchmark, an underpowered car needs to climb a hill (Sutton & Barto, 1998), and is described in Section B.1.3.

The experiments are designed to determine whether OAPI reliably minimizes the Bellman residual in comparison with API and ALP. We use a uniformly-spaced linear spline to approximate the value function. The constraints were based on 200 uniformly sampled states with all 3 actions per state. We evaluated the methods with the number of the approximation features 100 and 144, which corresponds to the number of linear segments.

The results of robust ABP (in particular OAPI), ALP, API with L_2 minimization, and LSPI are depicted in Table 5.1. The results are shown for both L_∞ norm and uniformly-weighted L_2 norm. The run-times of all these methods are comparable, with ALP being the fastest. Since API (LSPI) is not guaranteed to converge, we ran it for at most 20 iterations, which was an upper bound on the number of iterations of OAPI. The results demonstrate that ABP minimizes the L_∞ Bellman residual much more consistently than the other methods.

(a) L_∞ error of the Bellman residual

Features	100	144
OAPI	0.21 (0.23)	0.13 (0.1)
ALP	13. (13.)	3.6 (4.3)
LSPI	9. (14.)	3.9 (7.7)
API	0.46 (0.08)	0.86 (1.18)

(b) L_2 error of the Bellman residual

Features	100	144
OAPI	0.2 (0.3)	0.1 (1.9)
ALP	9.5 (18.)	0.3 (0.4)
LSPI	1.2 (1.5)	0.9 (0.1)
API	0.04 (0.01)	0.08 (0.08)

Table 5.1. Bellman residual of the final value function. The values are averages over 5 executions, with the standard deviations shown in parentheses.

Note, however, that all the considered algorithms would perform significantly better given a finer approximation.

5.6 Contributions

The main contributions described in this chapter are the bilinear formulations of the value function approximation and their analysis. In particular, we have shown that it is possible to minimize tight bounds on the policy loss. While solving this approximation is intractable (not in polynomial time), practical problems can be often solved in reasonable time. The bilinear formulation, however, does not add complexity, since solving the approximation is NP complete.

PART II

ALGORITHMS

CHAPTER 6

HOMOTOPY CONTINUATION METHOD FOR APPROXIMATE LINEAR PROGRAMS

Approximate linear programs are often very hard to solve because the number of constraints may be very large. A common problem with approximate dynamic programming is designing good features for the domain. One approach, which we describe in detail in Chapter 10, is to provide a large number of features and automatically choose the most suitable ones. This approach requires that regularized approximate linear programs with a large number of constraints and variables (variables correspond to constraints) are solved. This chapter proposes a homotopy continuation method for efficiently solving such linear programs.

We consider here regularized approximate linear programs. In these, the set of representable value functions is defined as:

$$\mathcal{M} = \{\Phi x \mid \|x\|_{1,e} \le \psi\},$$

where ψ is the regularization coefficient. The norm is weighted using a weight e. The homotopy continuation methods trace the optimal solution of the approximate linear program as a function of ψ. This can be efficient when the number of non-zero elements of x is small for small ψ. In addition, the homotopy method can be used to select the value of ψ, as described in Section 10.3.

The remainder of the chapter is organized as follows. First, Section 6.1 describes a basic homotopy continuation method that resembles the simplex method. This formulation can be problematic when the solutions of the linear program are degenerate. We propose an alternative approach in Section 6.2 based on penalty methods. Then, Section 6.3 describes a computationally efficient method for tracing the optimal solution. Section 6.5 describes the connections to other methods for solving regularized programs. Finally, Section 6.4 shows the efficiency of the homotopy method on some benchmark problems.

6.1 Homotopy Algorithm

This section presents a basic homotopy continuation method for solving regularized linear programs. The method exploits the sparsity of the solutions of regularized linear programs.

This algorithm can be seen as an extension of the standard *simplex algorithm* (Vanderbei, 2001). Simplex algorithm uses basic feasible solutions of the size of the number of variables, regardless of how many are non-zero. Because we are interested in solving very large linear programs, this is often infeasible. The homotopy method that we propose computes basic feasible solutions that correspond in size to the number of non-negative variables; it ignores the variables that are zero — typically the majority of them.

We consider the following formulation of the regularized approximate linear program (ALP):

$$\begin{aligned}\min_{x} \quad & c^{\mathsf{T}}\Phi x \\ \text{s.t.} \quad & A\Phi x \geq b \\ & e^{\mathsf{T}} x \leq \psi \\ & x \geq \mathbf{0}\end{aligned} \tag{ALP–R}$$

Remark 6.1. The variables in the linear program (ALP–R) are constrained to be non-negative ($x \geq \mathbf{0}$). This restriction does not impact generality but significantly simplifies the derivation of the homotopy algorithms; these constraints allow the L_1 norm to be represented as a single linear constraint. As a result, it is not necessary to use non-differentiable convex analysis (Osborne, Presnell, & Turlach, 2000), but the algorithm can be derived using standard convex optimization theory.

The methods that we propose in this chapter work for any generic linear program. Therefore, the remainder of the chapter considers general linear programs:

$$\begin{aligned}\min_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & A x \geq b \\ & e^{\mathsf{T}} x \leq \psi \\ & x \geq \mathbf{0}\end{aligned} \tag{LP–L_1}$$

This notation overloads symbols from the remainder of the thesis. For a matrix A, we use A_j to denote the j-th row and $A_{\cdot i}$ as i-the column. We also use $x(i)$ to denote the i-the element of the vector. The dual formulation of the linear program (LP–L_1) is:

$$\begin{aligned}\max_{y,\lambda} \quad & b^{\mathsf{T}} y - \psi\lambda \\ \text{s.t.} \quad & A^{\mathsf{T}} y - e\lambda \leq c \\ & y, \lambda \geq 0\end{aligned} \tag{6.1}$$

Definition 6.2. The optimal solution of the linear program (LP–L_1) as a function of ψ is denoted as $x : \mathbb{R} \to \mathbb{R}^{|\Phi|}$, assuming that the solution is a singleton. That is, $x(\psi)$ represents an optimal solution for the regularization coefficient ψ. This is the optimal solution, not the optimal objective value. We also use $y(\psi)$ and $\lambda(\psi)$ similarly to denote the sets of optimal solutions for the primal and dual programs respectively for the given the regularization coefficient.

The algorithm keeps a set of active variables $\mathcal{B}(x)$ and a set of active constraints $\mathcal{C}(x)$. A variable is considered to be active if it is non-zero. A constraint is considered to be active when the corresponding dual value is non-negative. Active and inactive variables and constraints are formally defined as follows:

$$\mathcal{B}(x) = \{i \,|\, x(i) > 0\} \quad \mathcal{N}(x) = \{i \,|\, x(i) = 0\} \quad \mathcal{C}(x) = \{j \,|\, y(j) > 0\} \quad \mathcal{D}(x) = \{j \,|\, y(j) = 0\}$$

When obvious, we drop the index x. That is we use $\mathcal{B}$ in place of $\mathcal{B}(x)$ when the value of x is apparent from the context.

Assumption 6.3. The optimal solution of the linear program (LP–L_1) is feasible and bounded for values of $\psi \in [0, \infty)$. In addition, it is "easy" to solve for $\psi = 0$.

The optimality conditions for the linear program (LP–L_1) can be either derived from KKT conditions (e.g. (Bertsekas, 2003; Boyd & Vandenberghe, 2004)), or from the complementary slackness optimality conditions (Vanderbei, 2001).

$$\begin{aligned} -e^{\mathsf{T}} x &\geq -\psi \\ Ax &\geq b \\ A^{\mathsf{T}} y &\leq c + \lambda e \\ y^{\mathsf{T}}(b - Ax) &= 0 \\ \lambda\ (e^{\mathsf{T}} x - \psi) &= 0 \\ x^{\mathsf{T}}(c - A^{\mathsf{T}} y + e\lambda) &= 0 \\ x, y, \lambda &\geq \mathbf{0} \end{aligned}$$

Using the active and inactive variables and constraints, the optimality equations can be reformulated as follows:

$$c = \begin{pmatrix} c_{\mathcal{B}} \\ c_{\mathcal{N}} \end{pmatrix} \qquad x = \begin{pmatrix} x_{\mathcal{B}} \\ x_{\mathcal{N}} \end{pmatrix} \qquad b = \begin{pmatrix} b_{\mathcal{C}} \\ b_{\mathcal{D}} \end{pmatrix} \qquad y = \begin{pmatrix} y_{\mathcal{C}} \\ y_{\mathcal{D}} \end{pmatrix} \qquad A = \begin{pmatrix} A_{\mathcal{BC}} & A_{\mathcal{NC}} \\ A_{\mathcal{BD}} & A_{\mathcal{ND}} \end{pmatrix}$$

Without loss of generality, we assume that the active constraints and variables are the first in the respective structures. Alternatively, this could be expressed generally using a permutation matrix P. We implicitly assume that the regularization vector e is partitioned properly for the active variables.

For a given set of active variables and constraints and using the fact that the inactive variables x and y are $\mathbf{0}$, the optimality conditions can be then written as:

$$\begin{aligned} & e^{\mathsf{T}} x_{\mathcal{B}} = \psi & & \\ & A_{\mathcal{BC}} x_{\mathcal{B}} = b_{\mathcal{C}} & & A_{\mathcal{BD}} x_{\mathcal{B}} \leq b_{\mathcal{D}} \\ & A_{\mathcal{BC}}^{\mathsf{T}} y_{\mathcal{C}} = c_{\mathcal{B}} + \lambda e_{\mathcal{B}} & & A_{\mathcal{NC}}^{\mathsf{T}} y_{\mathcal{C}} \leq c_{\mathcal{N}} + \lambda e_{\mathcal{N}} \\ & x, y, \lambda \geq \mathbf{0} & & \end{aligned}$$

The equality constraints are implied by the complementary slackness conditions and the fact that $x_{\mathcal{B}} > 0$ and $y_{\mathcal{C}} > 0$. We are assuming that the regularization constraint $\|x\|_{1,e} \leq \psi$ is active. If it becomes inactive at $\bar{\psi}$, the solution is optimal for any value of $\psi \geq \bar{\psi}$. The equalities follow from the complementarity conditions, omitted here. Notice that other constraints may also hold with equality.

Assumption 6.4. Only one constraint or variable enters the basis at the same time, and also at most one variable enters the program at the time.

The homotopy method is shown in algorithm 6.1. The primal update of the algorithm traces the solution in the linear segments and the dual update determines the direction to take in the non-linearities of the solution trace. The algorithm implementation is written with the focus on simplicity; an efficient implementation would use factorized matrices.

Algorithm 6.1: Homotopy Continuation Method for Solving ALP

```
$\psi^0 \leftarrow 0$ ;
// Find an initial feasible solutions
$x^0 \leftarrow x(\psi^0)$ ;
$y^0 \leftarrow y(\psi^0)$ ;
// Determine the initial active sets, and set $\mathcal{N}$ and $\mathcal{D}$ to be their complements
$\mathcal{B}^0 = \{i \mid x^0 > 0\}$ $\mathcal{C}^0 \leftarrow \{j \mid y(j) > 0\}$ // The regularization constraint is active, or the
    solution is optimal
while $\psi^i < \bar{\psi}$ and $\lambda^i > 0$ do
    $i \leftarrow i + 1$ ;
    // Here $|\mathcal{C}| = |\mathcal{B}| + 2$
    // Calculate the space (line) of dual solutions --- the update direction
    $\begin{pmatrix} \Delta y^i \\ \Delta \lambda^i \end{pmatrix} \leftarrow \text{null}\begin{pmatrix} A_{\mathcal{BC}}^{\mathsf{T}} & e \end{pmatrix}$ such that $y^{i-1}(\psi) = 0 \Rightarrow \Delta y^i(\psi) \geq 0$ ;  // This is always possible
    because there is always at most one such constraint, given the assumptions.
    // Decide based on a potential variable improvement
    // Calculate the maximum length of the update $\tau$, breaking ties arbitrarily.
    $t_1 \leftarrow \left(\max_{k \in \mathcal{C}} \frac{-\Delta y_k^i}{y_k^{i-1}}\right)^{-1}$ ;                // Some $y$ becomes 0.
    $t_2 \leftarrow \left(\max_{k \in \mathcal{N}} \frac{-\left(A_{\mathcal{NC}} \Delta y^i\right)_k}{\left(A_{\mathcal{NC}} y^{i-1}\right)_k}\right)^{-1}$ ;      // Some $x$ needs to be added to the active set.
    $t_3 \leftarrow \frac{\lambda}{-\Delta \lambda}$ ;                // Regularization constraint
    $\tau = \min\{t_1, t_2, t_3\}$ // Resolve the non-linearity update, where $K_l$ is the set of
        maximizers for $t_l$
    if $\tau = t_1$ then
        $\mathcal{C}^i \leftarrow \mathcal{C}^{i-1} \setminus K_1, \quad \mathcal{D}^i \leftarrow (\mathcal{C}^i)^{\mathsf{C}}$
    else if $\tau = t_2$ then
        $\mathcal{B}^i \leftarrow \mathcal{B}^{i-1} \cup K_2, \quad \mathcal{N}^i \leftarrow (\mathcal{B}^i)^{\mathsf{C}}$
    else if $\tau = t_3$ then
        The regularization constraint is inactive, return the solution.
    // Update the *dual* solutions
    $y^i \leftarrow y^{i-1} + \tau \Delta y^i, \quad \lambda^i \leftarrow \lambda^{i-1} + \tau \Delta \lambda^i$ ;
    // Here $|\mathcal{C}| = |\mathcal{B}| + 1$
    // Calculate the update direction
    $\Delta x \leftarrow \begin{pmatrix} A_{\mathcal{BC}} \\ e_{\mathcal{B}} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{0} \\ \Delta \psi \end{pmatrix}$ ;
    // Calculate the maximum length of the update $\tau$, breaking ties arbitrarily.
    $t_4 \leftarrow \left(\max_{k \in \mathcal{D}} \frac{-a_k^{\mathsf{T}} \Delta x^i}{a_k^{\mathsf{T}} x^{i-1} - b}\right)^{-1}$ ;                // A constraint becomes active
    $t_5 \leftarrow \left(\max_{k \in \mathcal{B}} \frac{-\Delta x_k^i}{a_k^{\mathsf{T}} x^{i-1} - b}\right)^{-1}$ ;                // A variable $\tau = \min\{t_4, t_5\}$
    // Update the *primal* solutions
    $x^i \leftarrow x_{i-1} + \tau \Delta x^i$ ;
    // Resolve the non-linearity update, where $K_l$ is the set of maximizers for $t_l$
    if $\tau = t_4$ then
        $\mathcal{C}^i \leftarrow \mathcal{C}^{i-1} \cup K_4, \quad \mathcal{D}^i \leftarrow (\mathcal{C}^i)^{\mathsf{C}}$
    else if $\tau = t_5$ then
        $\mathcal{B}^i \leftarrow \mathcal{B}^{i-1} \setminus K_5, \quad \mathcal{N}^i \leftarrow (\mathcal{B}^i)^{\mathsf{C}}$
```

The main idea of the homotopy method is to first calculate $\theta_L(0)$ and then trace the optimal solution for increasing values of ψ. The optimal solution $w(\psi)$ of the ALP is a piecewise linear function of ψ. At any point in time, the algorithm keeps track of a set of active – or non-zero – variables w and a set of active constraints, which are satisfied with equality. In the linear segments, the number of active constraints and variables are identical, and the non-linearity arises when variables and constraints become active or inactive. Therefore, the linear segments are traced until a variable becomes inactive variables or a constraint becomes active. Then, the dual solution is traced until a constraint becomes inactive or a variable becomes active.

The analysis above shows the following two properties of the homotopy method.

Proposition 6.5. *Given Assumption 6.4, the function $x(\psi)$ is piecewise linear. Algorithm 7 then converges to the optimal solution in a finite number of steps.*

The proof of the proposition can be found in Section C.7.

Remark 6.6. Relaxing the assumptions that exactly one constraint or variable are leaving or entering the problem. The solutions correspond to anti-cycling algorithms for solving linear programs.

6.2 Penalty-based Homotopy Algorithm

This section describes an alternative homotopy method for a penalty-based formulation (Mangasarian, 2004) of the linear program. This formulation simplifies the homotopy algorithm and makes it more efficient when dealing with degenerate solutions. In particular, the homotopy method on the regular ALP assumes a careful balance between the number of active constraints and the active variables. It is possible to address degenerate solutions using anti-cycling-like approaches for the simplex method (Vanderbei, 2001). These methods, however, are complicated and may need many steps to escape from degenerate solutions.

Degenerate primal solutions are caused by multiple feasible optimal solutions. Therefore, we address the problem with degenerate solutions by regularizing the dual solution. The dual program (LP–L_1):

$$\begin{aligned} \max_{y,\lambda} \quad & b^\mathsf{T} y - \psi\lambda \\ \text{s.t.} \quad & A^\mathsf{T} y - e\lambda \le c \\ & y, \lambda \ge 0 \end{aligned} \tag{6.2}$$

then becomes:

$$\begin{aligned} \max_{y,\lambda} \quad & b^\mathsf{T} y - \psi\lambda + \frac{1}{\chi} y^\mathsf{T} y \\ \text{s.t.} \quad & A^\mathsf{T} y - e\lambda \le c \\ & y, \lambda \ge 0 \end{aligned} \tag{6.3}$$

Note the regularization coefficient $y^\mathsf{T} y$. As we show below, it is possible to choose a sufficiently large coefficient χ without affecting the optimal solution. The corresponding primal formulation is:

$$\begin{aligned} \min_{x} \quad & f_2(x) = c^\mathsf{T} x + \chi \frac{1}{2} \left\| [Ax - b]_+ \right\|_2^2 \\ \text{s.t.} \quad & e^\mathsf{T} x \le \psi \\ & x \ge 0 \end{aligned} \tag{6.4}$$

This convex quadratic program corresponds to the quadratic penalty formulation of the linear program. It has been studied previously and it is often easier to solve than the original linear program (Mangasarian, 2004). The parameter χ represents the penalty for constraint violation — typically it is a large number.

Notice that feasible solutions in this formulation may violate the constraints for an arbitrarily large value of χ. However, it can be shown that there is a sufficiently large value χ that guarantees that the dual solution of the linear program (6.4) is the same as the dual solution of the linear program (LP–L_1).

Theorem 6.7. *For any linear program* (6.4) *and all values ψ, there exists a value $\chi > 0$ such that the optimal solution of the program* (6.4) *is also optimal in the linear program* (ALP–R)*.*

This is a straightforward extension of previous results (Pinar, 1996; Mangasarian, 2004); It follows by taking a maximal value over all values $\psi \in [0, \bar{\psi}]$. The proof uses the analysis of the dual formulation.

Given an optimal dual solution it is easy to obtain the primal optimal solution using the active constraints. Just as in the regular homotopy continuation method, we make the following assumption.

Definition 6.8. The optimal solution of the linear program (6.4) as a function of ψ is denoted as $x : \mathbb{R} \to \mathbb{R}^{|\Phi|}$, assuming that the solution is a singleton. Notice that this is the optimal solution, not the optimal objective value.

The homotopy method traces the trajectory of $x(\psi)$ starting with $\psi = 0$. That means that we have to guarantee that the values satisfy the optimality conditions at each point of the trajectory. As Proposition 6.9 shows, the trajectory of $x(\psi)$ is piecewise linear. Clearly, the function $f_2(x(\psi))$ is non-increasing in ψ.

To trace the trajectory of $x(\psi)$, we first write down the Lagrangian of the solution:

$$\begin{aligned} L(x, \lambda, \mu) &= c^\mathsf{T} x + \chi \frac{1}{2} \left\| [Ax - b]_+ \right\|_2^2 - \lambda(\psi - e^\mathsf{T} x) - x^\mathsf{T} \mu \\ &= c^\mathsf{T} x + \chi \frac{1}{2} \left(A_{\cdot \mathcal{C}} x - b_{\mathcal{C}} \right)^2 - \lambda(\psi - e^\mathsf{T} x) - x^\mathsf{T} \mu \\ &= c^\mathsf{T} x + \chi \frac{1}{2} \left(x^\mathsf{T} A_{\cdot \mathcal{C}}^\mathsf{T} A_{\cdot \mathcal{C}} x - 2 b_{\mathcal{C}}^\mathsf{T} A_{\cdot \mathcal{C}} x + b_{\mathcal{C}}^\mathsf{T} b_{\mathcal{C}} \right) - \lambda(\psi - e^\mathsf{T} x) - x^\mathsf{T} \mu \end{aligned}$$

Here λ is the dual variable that correspond to the constraint $e^\mathsf{T} x \leq \psi$ and μ_i are the dual variables corresponding to constraints $x_i \geq 0$.

To take advantage of the sparsity of the solutions, we define active and inactive constraints and variables. In general, the inactive constraints and variables can be largely ignored when tracing the optimal solution in linear segments. The active variables and constraints are defined as follows:

Active variables All non-zero variables and potentially and variable that may be non-zero in adjacent segments. These are variables that are non-zero currently adjacent linear segments of non-linearity points.

$$\{i \,|\, x_i > 0\} \subseteq \mathcal{B} \subseteq \{i \,|\; \mu_i = 0\} \tag{6.5}$$

Inactive variables Complement of active variables.

$$\mathcal{N} = \mathcal{B}^{\mathsf{C}} \tag{6.6}$$

Active constraints All violated constraints and some exactly-satisfied constraints. These are constraints that may be violated currently or in adjacent linear segments of nonlinearity points.

$$\{j \,|\, a_j^{\mathsf{T}} x < b_j\} \subseteq \mathcal{C} \subseteq \{j \,|\, a_j^{\mathsf{T}} x \leq b_j\} \tag{6.7}$$

Inactive constraints Complement of active constraints.

$$\mathcal{D} = \mathcal{C}^{\mathsf{C}} \tag{6.8}$$

The homotopy algorithm in Section 6.1 uses dual variables instead to define the active constraints. A similar definition would be possible also in this case, but we do not provide it for the sake of simplicity. The unrestricted sets of all constraints or variables are represented using "$\cdot$".

For a given set of active and inactive variables and constraints, the Lagrangian can be written as:

$$\begin{aligned} L(x,\lambda,\mu) &= c^{\mathsf{T}} x + \chi \frac{1}{2} \left(x^{\mathsf{T}} A_{\cdot\mathcal{C}}^{\mathsf{T}} A_{\cdot\mathcal{C}} x - 2 b_{\mathcal{C}}^{\mathsf{T}} A_{\cdot\mathcal{C}} x + b_{\mathcal{C}}^{\mathsf{T}} b_{\mathcal{C}} \right) - \lambda(\psi - e^{\mathsf{T}} x) - x^{\mathsf{T}} \mu \\ &= \chi \frac{1}{2} \left(x_{\mathcal{B}}^{\mathsf{T}} A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}} + x_{\mathcal{N}}^{\mathsf{T}} A_{\mathcal{NC}}^{\mathsf{T}} A_{\mathcal{NC}} x_{\mathcal{N}} + \right. \\ &\qquad \left. + 2 x_{\mathcal{N}}^{\mathsf{T}} A_{\mathcal{NC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}} - 2 b_{\mathcal{C}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}} - 2 b_{\mathcal{C}}^{\mathsf{T}} A_{\mathcal{NC}} x_{\mathcal{N}} + b_{\mathcal{C}}^{\mathsf{T}} b_{\mathcal{C}} \right) + \\ &\qquad + c_{\mathcal{B}}^{\mathsf{T}} x_{\mathcal{B}} + c_{\mathcal{N}}^{\mathsf{T}} x_{\mathcal{N}} - \lambda(\psi - e_{\mathcal{B}}^{\mathsf{T}} x_{\mathcal{B}} - e_{\mathcal{N}}^{\mathsf{T}} x_{\mathcal{N}}) - x_{\mathcal{B}}^{\mathsf{T}} \mu_{\mathcal{B}} \end{aligned}$$

The optimality conditions are then:

$$\nabla_{x_{\mathcal{B}}} L = c_{\mathcal{B}} - \chi A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} + \chi A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}} + \lambda e_{\mathcal{B}} - \mu_{\mathcal{B}} = \mathbf{0} \tag{6.9}$$

$$\nabla_{x_{\mathcal{N}}} L = c_{\mathcal{N}} - \chi A_{\mathcal{NC}}^{\mathsf{T}} b_{\mathcal{C}} + \chi A_{\mathcal{NC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}} + \lambda e_{\mathcal{N}} - \mu_{\mathcal{N}} = \mathbf{0} \tag{6.10}$$

Additionally, the optimal solution must also satisfy the following constraints:

$$\begin{array}{llll} x_{\mathcal{B}} & \geq \mathbf{0} & x_{\mathcal{N}} & = \mathbf{0} \\ \mu_{\mathcal{B}} & = \mathbf{0} & \mu_{\mathcal{N}} & \geq \mathbf{0} \\ A_{\cdot\mathcal{C}} x & \leq b_{\mathcal{C}} & A_{\cdot\mathcal{D}} x & > b_{\mathcal{D}} \\ \lambda & \geq 0 & & \end{array} \tag{6.11}$$

The optimal values of all variables are parameterized as functions of ψ. The optimality condition for $x_{\mathcal{B}}(\psi)$ can be calculated from the equations (6.9) as follows:

$$\begin{aligned} \chi A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}}(\psi) &= -\lambda(\psi) e_{\mathcal{B}} + \chi A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} - c_{\mathcal{B}} \\ x_{\mathcal{B}}(\psi) &= \left(\chi A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} \left(-e_{\mathcal{B}} \lambda(\psi) + \chi A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} - c_{\mathcal{B}} \right) \\ x_{\mathcal{B}}(\psi) &= -\left(\chi A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} \left(e_{\mathcal{B}} \lambda(\psi) + c_{\mathcal{B}} \right) + \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} \end{aligned} \tag{6.12}$$

Assuming that the constraint $e^{\mathsf{T}} x \leq \psi$ is satisfied with equality (Vanderbei, 2001), the value of $\lambda(\psi)$ can be computed from the equations (6.12) as follows:

$$\begin{aligned}
\psi &= e_{\mathcal{B}}^{\mathsf{T}} x_{\mathcal{B}} \\
\chi\psi &= -e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} \left(e_{\mathcal{B}}\lambda(\psi) + c_{\mathcal{B}}\right) + \chi e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} \\
\lambda(\psi) e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} e_{\mathcal{B}} &= e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} \left(\chi A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} - c_{\mathcal{B}}\right) - \chi\psi \\
\lambda(\psi) &= \frac{e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} \left(\chi A_{\mathcal{BC}}^{\mathsf{T}} b_{\mathcal{C}} - c_{\mathcal{B}}\right) - \chi\psi}{e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} e_{\mathcal{B}}} && (6.13)
\end{aligned}$$

Note that $x_{\mathcal{B}}$ and λ can be computed simultaneously by combining (6.12) and (6.13). This is more practical to compute, but we show separate derivations to simplify the proofs of the properties of the algorithm.

The variables $\mu_{\mathcal{N}}$ can be calculated from (6.10) as follows:

$$\mu_{\mathcal{N}}(\psi) = c_{\mathcal{N}} - \chi A_{\mathcal{NC}}^{\mathsf{T}} b_{\mathcal{C}} + \chi A_{\mathcal{NC}}^{\mathsf{T}} A_{\mathcal{BC}} x_{\mathcal{B}}(\psi) + e_{\mathcal{N}}\lambda(\psi) \tag{6.14}$$

The difference between values of the regularization coefficient is $\Delta\psi = \psi' - \psi$. Then, the differences in optimal values of other variables are calculated as follows:

$$\begin{aligned}
\Delta\lambda = \lambda(\psi') - \lambda(\psi) &= \frac{-\chi\Delta\psi}{e_{\mathcal{B}}^{\mathsf{T}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} e_{\mathcal{B}}} && (6.15) \\
\Delta x_{\mathcal{B}} = x(\psi') - x(\psi) &= -\left(\chi A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}}\right)^{-1} e_{\mathcal{B}} \Delta\lambda && (6.16) \\
\Delta\mu_{\mathcal{N}} = \mu_{\mathcal{N}}(\psi') - \mu_{\mathcal{N}}(\psi) &= \chi A_{\mathcal{NC}}^{\mathsf{T}} A_{\mathcal{BC}} \Delta x_{\mathcal{B}} + e_{\mathcal{N}}\Delta\lambda && (6.17)
\end{aligned}$$

In addition, $\Delta x_{\mathcal{N}} = \mathbf{0}$ and $\Delta\mu_{\mathcal{B}} = \mathbf{0}$. Notice that the update directions of all variables are linear in $\Delta\psi$. The solution is linear an can be traced as long as the variables satisfy the equalities in (6.11). At that point it may be necessary to update the active sets of variables and constraints. The detailed update described in algorithm 6.2. The derivation above can be summarized by the following proposition.

Proposition 6.9. *The optimal solution $x(\psi)$ of the linear program* (6.4) *with respect to ψ is a piecewise linear function.*

Theorem 6.10. *algorithm 6.2 converges to the optimal solution in a finite number of steps.*

The proof of the theorem can be found in Section C.7.

6.3 Efficient Implementation

This section shows a basic approach for efficiently updating the matrices involved in the penalty-based homotopy method. This is a basic approach and better results may be obtained using an LU decomposition, similarly to the standard simplex method.

Algorithm 6.2: Homotopy method for (6.4).

```
// Set the initial regularization parameter $\psi$
$\psi^0 \leftarrow 0$ ;
// Calculate an initial solutions (Easy to calculate)
$x^0 \leftarrow x(0)$ ;                                                        // (6.4)
$\lambda^0 \leftarrow \lambda(0)$ ;                                            // (6.13)
$\mu^0 \leftarrow \mu(0)$;                                                     // (6.14)
Initialize, maximizing the active sets: $\mathcal{B}^0, \mathcal{N}^0, \mathcal{C}^0, \mathcal{D}^0$ ;   // (6.5),(6.6),(6.7),(6.8)
// Loop through non-linearity points while the regularization
   constraint is active
$i \leftarrow 0$ ;                                                             // Counter
while $\lambda^i > 0$ do
    $i \leftarrow i + 1$ ;
    Calculate update directions: $\Delta x^i$, $\Delta\lambda^i$, $\Delta\mu^i$ ;   // (6.16),(6.15),(6.17)
    // Calculate the maximal length of the update $\tau$, breaking ties
       arbitrarily.
    $t_1 \leftarrow \left(\max_{k\in\mathcal{N}} \frac{-\Delta\mu_k^i}{\mu_k^{i-1}}\right)^{-1}$   $t_2 \leftarrow \left(\max_{k\in\mathcal{B}} \frac{-\Delta x_k^i}{x_k^{i-1}}\right)^{-1}$ ;   // Variable change
    $t_3 \leftarrow \left(\max_{k\in\mathcal{D}} \frac{-a_k^\intercal \Delta x^i}{a_k^\intercal x^{i-1} - b}\right)^{-1}$   $t_4 \leftarrow \left(\max_{k\in\mathcal{C}} \frac{-a_k^\intercal \Delta x^i}{a_k^\intercal x^{i-1} - b}\right)^{-1}$ ;   // Constraint
    change
    $t_5 \leftarrow \frac{\lambda}{-\Delta\lambda}$ ;                          // Regularization constraint
    $\tau = \min\{t_1, t_2, t_3, t_4, t_5\}$ ;
    // Update solutions
    $x^i \leftarrow x^{i-1} + \tau\Delta x^i$,   $\lambda^i \leftarrow \lambda^{i-1} + \tau\Delta\lambda^i$,   $\mu^i \leftarrow \mu^{i-1} + \tau\Delta\mu^i$ ;
    // Resolve the non-linearity update, where $K_l$ is the set of
       maximizers for $t_l$
    if $\tau = t_1$ then
        $\mathcal{B}^i \leftarrow \mathcal{B}^{i-1} \cup K_1$,   $\mathcal{N}^i \leftarrow (\mathcal{B}^i)^\complement$
    else if $\tau = t_2$ then
        $\mathcal{B}^i \leftarrow \mathcal{B}^{i-1} \setminus K_2$,   $\mathcal{N}^i \leftarrow (\mathcal{B}^i)^\complement$
    else if $\tau = t_3$ then
        $\mathcal{C}^i \leftarrow \mathcal{C}^{i-1} \setminus K_3$,   $\mathcal{D}^i \leftarrow (\mathcal{C}^i)^\complement$
    else if $\tau = t_4$ then
        $\mathcal{C}^i \leftarrow \mathcal{C}^{i-1} \cup K_4$,   $\mathcal{D}^i \leftarrow (\mathcal{C}^i)^\complement$
    else if $\tau = t_5$ then
        The regularization constraint is inactive, return the solution.
```

Sherman-Morrison-Woodbury formula

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1} \tag{6.18}$$

Schur complement

$$\begin{aligned}
M &= \begin{pmatrix} A & B \\ C & D \end{pmatrix} \\
S &= A - BD^{-1}C \qquad (6.19)\\
M^{-1} &= \begin{pmatrix} S^{-1} & -S^{-1}B\,D^{-1} \\ -D^{-1}CS^{-1} & D^{-1}+D^{-1}CS^{-1}BD^{-1} \end{pmatrix} \qquad (6.20)
\end{aligned}$$

Inverse of Schur complement

$$S = \left(A - BD^{-1}C\right)^{-1} = A^{-1} - A^{-1}B\left(D + CA^{-1}B\right)^{-1}CA^{-1} \tag{6.21}$$

The inverse matrix that needs to be calculated in algorithm 6.2 is $\left(A_{\mathcal{BC}}^{\mathsf{T}}A_{\mathcal{BC}}\right)^{-1}$. The update of the matrix in various situations will be:

Adding active variables The new matrix is denoted as $\bar{A}$ and the columns that correspond to the additional variable are Q.

$$\begin{aligned}
\bar{A} &= \begin{pmatrix} A & Q \end{pmatrix} \\
\bar{A}^{\mathsf{T}}A &= \begin{pmatrix} A^{\mathsf{T}} \\ Q^{\mathsf{T}} \end{pmatrix}\begin{pmatrix} A & Q \end{pmatrix} = \begin{pmatrix} A^{\mathsf{T}}A & AQ \\ Q^{\mathsf{T}}A & Q^{\mathsf{T}}Q \end{pmatrix} \\
\bar{S} &= A^{\mathsf{T}}A - AQ\left(Q^{\mathsf{T}}Q\right)^{-1}Q^{\mathsf{T}}A \\
\bar{S}^{-1} &= \left(A^{\mathsf{T}}A\right)^{-1} - \left(A^{\mathsf{T}}A\right)^{-1}AQ\left(Q^{\mathsf{T}}Q + Q^{\mathsf{T}}A\left(A^{\mathsf{T}}A\right)^{-1}AQ\right)^{-1}Q^{\mathsf{T}}A\left(A^{\mathsf{T}}A\right)^{-1} \qquad (6.22)\\
\left(\bar{A}^{\mathsf{T}}A\right)^{-1} &= \begin{pmatrix} \bar{S}^{-1} & -\bar{S}^{-1}AQ\left(Q^{\mathsf{T}}Q\right)^{-1} \\ -\left(Q^{\mathsf{T}}Q\right)^{-1}Q^{\mathsf{T}}A\bar{S}^{-1} & \left(Q^{\mathsf{T}}Q\right)^{-1} + \left(Q^{\mathsf{T}}Q\right)^{-1}Q^{\mathsf{T}}A\bar{S}^{-1}AQ\left(Q^{\mathsf{T}}Q\right)^{-1} \end{pmatrix} \qquad (6.23)
\end{aligned}$$

Removing active variables Simply a reverse of adding variables.

Adding active constraints The new matrix is denoted as $\bar{A}$ and the columns that correspond to the additional variable are Q.

$$\begin{aligned}
\bar{A} &= \begin{pmatrix} A \\ Q \end{pmatrix} \\
\bar{A}^{\mathsf{T}}A &= \begin{pmatrix} A^{\mathsf{T}} & Q^{\mathsf{T}} \end{pmatrix}\begin{pmatrix} A \\ Q \end{pmatrix} = A^{\mathsf{T}}A + QQ^{\mathsf{T}} \qquad (6.24)\\
\left(\bar{A}^{\mathsf{T}}A\right)^{-1} &= \left(A^{\mathsf{T}}A\right)^{-1} - \left(A^{\mathsf{T}}A\right)^{-1}Q\left(I + Q^{\mathsf{T}}\left(A^{\mathsf{T}}A\right)^{-1}Q\right)^{-1}Q^{\mathsf{T}}\left(A^{\mathsf{T}}A\right)^{-1} - \qquad (6.25)\\
&\quad - \left(A^{\mathsf{T}}A\right)^{-1} \qquad (6.26)
\end{aligned}$$

Removing active constraints Simply a reverse of adding constraints.

6.4 Empirical Evaluation

We evaluate the empirical performance of the homotopy methods on the mountain car benchmark problem, described in more detail in Section B.1.3. The main utility of the homotopy method is that it can be used to select the value of ψ, as described in Section 10.3.

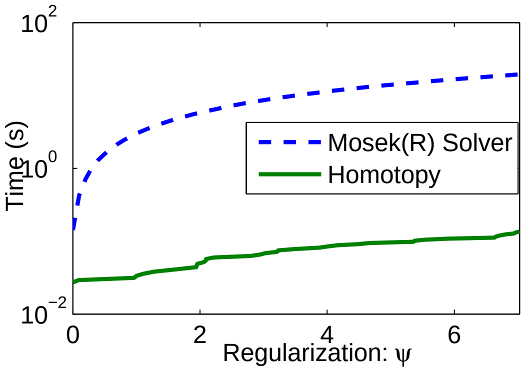

Figure 6.1. Comparison of performance of homotopy method versus Mosek as a function of ψ in the mountain car domain. The Mosek solutions are recomputed in increments for values of ψ.

As we show here, it may, however, perform better than commercial linear program solvers when the coefficients of the approximate solution are sparse.

We evaluate the relaxed homotopy method, because it is easier to implement. This implementation does not suffer when a large number of constants become active. With the right tie–breaking, the regular homotopy method is likely to perform identically.

The result on the mountain car domain show that the homotopy method performs significantly faster than a commercially available linear program solver Mosek. Figure 6.1 compares the computational time of homotopy method and Mosek, when solving the problem for multiple values of ψ in increments of 0.5 on the standard mountain car problem (Sutton & Barto, 1998) with 901 piecewise linear features and 6000 samples. Even for any *single* value ψ, the homotopy method solves the linear program about 3 times faster than Mosek, as Figure 6.2 illustrates.

The performance of the homotopy method depends significantly on the structure of the problem and is not likely to outperform general linear program solver for a fixed value of ψ. More understanding of how the properties of the problem domain influence the solution sparsity will be necessary to understand when using a homotopy method may be advantageous. However, the strength of the homotopy method is that it computes the optimal solution for a range of value ψ, which can be used to select its proper value, as Section 10.3 shows.

6.5 Discussion and Related Work

Regularization using the L_1 norm has recently gained popularity in solving regression problems. The goal in regression is to approximate a function from samples, which is a much simpler setting than value function approximation. The L_1 norm encourages sparsity, unlike the more standard L_2 norm. That is, the optimal solutions tend to have only a small number of non-zero coefficients. The practical consequences are the lower computational requirements and classifiers that are easier to use. The second motivation is that the L_1

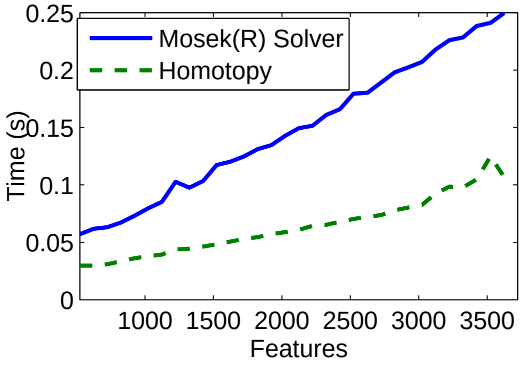

Figure 6.2. Comparison of performance of homotopy method versus Mosek as a function of ψ in the mountain car domain. The Mosek solutions are computed for the final value of ψ only.

regularization problem is closely related to Dantzig selector (Candes & Tao, 2007), which was proposed to solve regression problems:

$$\begin{aligned} \min_{\beta} \quad & \|\beta\|_1 \\ \text{s.t.} \quad & \|X^*(y - X\beta)\|_\infty \leq (1 + t^{-1})\sqrt{2\log p}\sigma \end{aligned} \tag{6.27}$$

This formulation is guaranteed to find a representation close to the true representation β^*, even when the number of features is greater than the number of samples. This requires some technical assumptions that roughly correspond to the features being close to orthogonal.

Because of the similarity of the Dantzig selector to L_1 regularized ALPs, it is interesting to discuss the solution methods for Dantzig selector. The initial work proposed to use a modification of an interior-point method for this purpose (Candes & Tao, 2007). However, (Friedlander & Saunders, 2007) argues that the computational complexity of solving for the Dantzig selector may be much higher than solving the related LASSO optimization problem (Hastie, Tibshirani, & Friedman, 2009).

LARS is a popular method for solving LASSO regularized problems by using a homotopy continuation method. It is efficient when the optimal solution is sparse — that is it has only a small number of positive entries. To replicate the success of LARS, homotopy continuation methods have recently been developed for Dantzig selector. DASSO (James, Radchenko, & Lv, 2009) and Primal dual pursuit (Asif, 2008, 2009) are some notable ones. Both methods essentially implement the same algorithm. These methods are in spirit similar to the Simplex-like homotopy method that we propose, but are specifically tailored to solving the Dantzig selector and are thus unsuitable for solving ALPs.

A connection between solving parametric linear programs and solving regularized linear programs, such as regularized ALP, is described in (Yao & Lee, 2007). The authors propose to use a parametric simplex algorithm for computing the whole homotopy path. The parametric simplex algorithm, however, does not take advantage of the potential sparsity of the solutions for a low value of the regularization parameter. Therefore it is not suitable for our purpose. The algorithm we propose could be seen as a sparse modification of the simplex algorithm.

6.6 Contributions

The homotopy method presented in this chapter is new. The basic homotopy method is closely related to DASSO and similar homotopy methods and does not represent the significant contribution. The main contribution in this chapter is the new simplified homotopy method for solving linear programs, which is based on a quadratic penalty formulation. This method is easier to implement than the basic one, and does not suffer from problems when the solutions are highly degenerate. In addition, it corresponds to formulations that have been extensively studied previously.

CHAPTER 7

SOLVING APPROXIMATE BILINEAR PROGRAMS

This chapter studies methods for solving approximate bilinear programs. Bilinear programs are an example of a global optimization problem because they represent minimization of a concave function. Because global optimization problems are usually intractable, it is important to study methods that can solve them approximately. Here, we describe precise and approximate methods for solving general bilinear programs and show how the structure of ABPs can be used to derive simpler formulations and methods.

Most of the existing approaches for solving bilinear programs focus on optimal solutions to relatively small bilinear programs. ABPs are, on the other hand, usually large and approximate solutions are sufficient, since the optimal solution is an approximation of the optimal value function anyway. This chapter proposes new methods that can be used to approximately compute ABPs and describes why the more traditional methods are impractical.

The chapter is organized as follows. First, Section 7.1 describes the basic approaches to solving bilinear programs. This elaborates on the brief discussion in Chapter 5 and also describes the most common method for solving bilinear programs, which is based on concavity cuts. Section 7.2 shows how general bilinear programs can be formulated as mixed integer linear programs — the most studied global optimization problem formulation. This formulation is, however, cumbersome and Section 7.3 proposes an ABP-specific formulation. Section 7.4 studies the properties of a generic homotopy method for regularized approximate bilinear programs. These properties are necessary to enable feature selection, discussed in Chapter 10.

7.1 Solution Approaches

Solving a bilinear program is an NP-complete problem (Bennett & Mangasarian, 1992). The membership in NP follows from the finite number of basic feasible solutions of the individual linear programs, each of which can be checked in polynomial time. The NP-hardness is shown by a reduction from the SAT problem (Bennett & Mangasarian, 1992).

Bilinear programs are non-convex and are typically solved using global optimization techniques. The common solution methods are based on concave cuts (Horst & Tuy, 1996), branch-and-bound (Carpara & Monaci, 2009), and successive approximation (Petrik & Zilberstein, 2007a). There are, however, no commercial solvers available for solving bilinear programs. Bilinear programs can be formulated as concave quadratic minimization problems (Horst & Tuy, 1996), or mixed integer linear programs (Horst & Tuy, 1996; Petrik & Zilberstein, 2007b), for which there are numerous commercial solvers available. Because we are interested in solving very large bilinear programs, we describe simple approximate algorithms next.

There are two main approaches to solving bilinear programs. In the first approach, a relaxation of the linear program is solved. The solution of the relaxed problem represents a lower bound on the optimal solution. The relaxation is then iteratively refined until the solution becomes feasible. In the second approach, feasible solutions of the bilinear program are calculated. These are successively refined to calculate optimal solutions. An example of such a method is based on *concavity cuts*.

Concavity cuts (Horst & Tuy, 1996) are the most common method for solving bilinear programs. These are cuts that are added between the iterations of algorithm 5.1. They eliminate the computed local solutions. New iterations of the iterative algorithm may then discover better solutions. While concavity cuts can be applied to solve approximate bilinear programs, it is not always practical to use them. Computing a cut requires solving a linear program multiple times. The number of linear programs that need to be solved corresponds to the number of samples in ABP and each of the linear programs is about as large as an ALP. This may be quite time consuming and the number of cuts required to achieve the optimality may be very large.

The solution methods that rely on the relaxed formulation are typically based on methods for solving integer linear programs. This is possible because bilinear programs can be easy translated to integer linear programs, as shown in Sections 7.2 and 7.3. The relaxation of the bilinear program is either a linear or semi-definite program. This relaxation is used either to calculate lower bounds in the branch and bound method or calculate suboptimal solutions. The relaxations are progressively refined using cutting plane methods.

The solution methods that refine locally optimal solutions of bilinear programs are based on algorithm 5.1. This algorithm iteratively finds a local optimum and adds cuts that eliminate the current suboptimal solution. Therefore, when a local search algorithm is executed again on the bilinear program a different locally optimal solution is found. This locally optimal solution is guaranteed to be different — albeit not better — than the previous one. The algorithm stops when the set of feasible solutions is empty.

7.2 General Mixed Integer Linear Program Formulation

This section shows how to formulate a general bilinear program as a mixed integer linear program. The formulation we derive here can be seen in a sense as a dual formulation of the bilinear program. The formulation relies on the dual variables of the individual linear programs, but is not truly dual since it is not a bilinear program.

This section shows a derivation for the following *general* bilinear program:

$$\begin{aligned} \min_{w,x \mid y,z} \quad & s_1^\mathsf{T} w + r_1^\mathsf{T} x + x^\mathsf{T} C y + r_2^\mathsf{T} y + s_2^\mathsf{T} z \\ \text{s.t.} \quad & A_1 x + B_1 w \geq b_1 \qquad\qquad A_2 y + B_2 z \geq b_2 \end{aligned} \tag{7.1}$$

This formulation is not in the normal form (see Chapter 8) and is not directly related to any ABP representation; it is chosen to simplify the derivation. This general formulation overloads some symbols from the remainder of the thesis. In particular, the variables λ introduced below are unrelated to the identically named variables in approximate bilinear programs.

The bilinear program consists of two linear programs:

$$\begin{array}{ll} \min\limits_{w,x} & s_1^\mathsf{T} + r_1^\mathsf{T} + x^\mathsf{T} C y \\ \text{s.t.} & A_1 x + B_1 w \geq b_1 \end{array}$$

and

$$\begin{array}{ll} \min\limits_{y,z} & x^\mathsf{T} C y + r_2^\mathsf{T} y + s_2^\mathsf{T} z \\ \text{s.t.} & A_2 + B_2 z \geq b_2 \end{array}$$

An optimal solution of (7.1) must be also optimal in the individual linear programs. Otherwise, the objective function of the bilinear program can be improved by improving the objective function of one of the linear programs. As a result, the necessary conditions for the optimality of a solution of (7.1) are:

$$\begin{aligned} A_1^\mathsf{T} \lambda_1 &= r_1 + C y & A_2^\mathsf{T} \lambda_2 &= r_2 + C^\mathsf{T} x \\ B_1^\mathsf{T} \lambda_1 &= s_1 & B_2^\mathsf{T} \lambda_2 &= s_2 \\ \lambda_1^\mathsf{T} (A_1 x + B_1 w - b_1) &= \mathbf{0} & \lambda_2^\mathsf{T} (A_2 y + B_2 z - b_2) &= \mathbf{0} \\ \lambda_1 &\geq \mathbf{0} & \lambda_2 &\geq \mathbf{0} \end{aligned}$$

These necessary optimality conditions can be derived either using complementary slackness conditions (Vanderbei, 2001) or Karush-Kuhn-Tucker conditions (Boyd & Vandenberghe, 2004).

Using the inequalities above, the optimal solution of the bilinear program is expressed as:

$$\begin{aligned} s_1^\mathsf{T} w + r_1^\mathsf{T} x + x^\mathsf{T} C y + r_2^\mathsf{T} y + s_2^\mathsf{T} z &= s_1^\mathsf{T} w + x^\mathsf{T} (C y + r_1) + r_2^\mathsf{T} y + s_2^\mathsf{T} z \\ &= s_1^\mathsf{T} w + s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 - \lambda_1^\mathsf{T} B_1 w \\ &= s_1^\mathsf{T} w + s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 - s_1^\mathsf{T} w \\ &= s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 \end{aligned}$$

This leads to the following optimization problem:

$$\begin{array}{lll} \min\limits_{w,x,\lambda_1 \mid y,z,\lambda_2} & s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 & \\ \text{s.t.} & A_1 x + B_1 w \geq b_1 & A_2 + B_2 z \geq b_2 \\ & A_1^\mathsf{T} \lambda_1 = r_1 + C y & A_2^\mathsf{T} \lambda_2 = r_2 + C^\mathsf{T} x \\ & B_1^\mathsf{T} \lambda_1 = s_1 & B_2^\mathsf{T} \lambda_2 = s_2 \\ & \lambda_1^\mathsf{T} (A_1 x + B_1 w - b_1) = \mathbf{0} & \lambda_2^\mathsf{T} (A_2 y + B_2 z - b_2) = \mathbf{0} \\ & \lambda_1 \geq \mathbf{0} & \lambda_2 \geq \mathbf{0} \end{array}$$

The program is linear with the exception of the complementarity constraints that involve λ_1^T and λ_2^T. In the special case when $C = \mathbf{I}$, the original variables can be replaced by $y = A_1^\mathsf{T} \lambda_1 - r_1$ and $x = A_2^\mathsf{T} \lambda_1 - r_2$ leading to the following:

$$
\begin{array}{lll}
\min\limits_{w,\lambda_1 \,|\, z,\lambda_2} & s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 & \\
\text{s.t.} & A_1 A_2^\mathsf{T} \lambda_2 - A_1 r_2 + B_1 w \geq b_1 & A_2 A_1^\mathsf{T} \lambda_1 - A_2 r_1 + B_2 z \geq b_2 \\
& A_1^\mathsf{T} \lambda_1 = r_1 + C y & A_2^\mathsf{T} \lambda_2 = r_2 + C^\mathsf{T} x \\
& B_1^\mathsf{T} \lambda_1 = s_1 & B_2^\mathsf{T} \lambda_2 = s_2 \\
& \lambda_1^\mathsf{T} A_1 A_2^\mathsf{T} \lambda_2 - \lambda_1^\mathsf{T} A_1 r_2 + \lambda_1^\mathsf{T} B_1 w - \lambda_1^\mathsf{T} b_1 = \mathbf{0} & \\
& \lambda_2^\mathsf{T} A_2 A_1^\mathsf{T} \lambda_1 - \lambda_2^\mathsf{T} A_2 r_1 + \lambda_2^\mathsf{T} B_2 z - \lambda_2^\mathsf{T} b_2 = \mathbf{0} & \\
& \lambda_1 \geq \mathbf{0} & \lambda_2 \geq \mathbf{0}
\end{array}
$$

This is not a linear program, since it contains complementarity constraints, such as

$$\lambda_2^\mathsf{T} A_2 A_1^\mathsf{T} \lambda_1 - \lambda_2^\mathsf{T} A_2 r_1 + \lambda_2^\mathsf{T} B_2 z - \lambda_2^\mathsf{T} b_2 = \mathbf{0}.$$

It is not a separable bilinear program either, because the bilinear terms are now in the constraints, not the objective function. However, the complementarity constraints can be easily linearized using a set of integer variables. In particular, $xy = 0$ for non-negative x and y can be replaced by

$$x \leq M(1-q) \qquad y \leq Mq$$

for $q \in \{0,1\}$ and some sufficiently large constant M. The mixed integer linear program is then:

$$
\begin{array}{lll}
\min\limits_{w,\lambda_1,z,\lambda_2,q_1,q_2} & s_2^\mathsf{T} z + \lambda_1^\mathsf{T} b_1 & \\
\text{s.t.} & A_1 A_2^\mathsf{T} \lambda_2 - A_1 r_2 + B_1 w \geq b_1 & A_2 A_1^\mathsf{T} \lambda_1 - A_2 r_1 + B_2 z \geq b_2 \\
& A_1^\mathsf{T} \lambda_1 = r_1 + C y & A_2^\mathsf{T} \lambda_2 = r_2 + C^\mathsf{T} x \\
& B_1^\mathsf{T} \lambda_1 = s_1 & B_2^\mathsf{T} \lambda_2 = s_2 \\
& A_1 A_2^\mathsf{T} \lambda_2 - A_1 r_2 + B_1 w - b_1 \leq (1-q_1) M & \\
& A_2 A_1^\mathsf{T} \lambda_1 - A_2 r_1 + B_2 z - b_2 \leq (1-q_2) M & \\
& \lambda_1 \leq M q_1 & \lambda_2 \leq M q_2 \\
& \lambda_1 \geq \mathbf{0} & \lambda_2 \geq \mathbf{0} \\
& q_1 \in \{0,1\}^n & q_2 \in \{0,1\}^n
\end{array}
\tag{7.2}
$$

Here M needs to be an upper bound (scalar) on λ_1 and λ_2. While such a bound may be difficult to estimate, any very large constant works in practice.

There are two main difficulties with this mixed integer linear program formulation. First, it is very large. Even though the increase in size is only linear in comparison with the bilinear program, this is significant because the approximate bilinear programs tend to be large. The number of integer variables is also large. Second, the structure of this mixed integer linear program is quite complex, making it hard to exploit the specific the structure of the approximate bilinear programs.

7.3 ABP-Specific Mixed Integer Linear Program Formulation

The formulation in Section 7.2 is general but very impractical for approximate bilinear programs. In this section, we present a more compact and structured mixed integer linear program formulation, which relies on the specific properties of approximate bilinear programs. In particular, it uses the property that there is always a solution with an optimal deterministic policy (see Theorem 5.2).

As in other parts of the thesis, we only show a formulation of the robust bilinear program (ABP–L_∞); the same approach applies to all other formulations that we propose. For ease of reference, the robust approximate bilinear program is:

$$\begin{aligned}
\min_{\pi \,|\, \lambda,\lambda',v} \quad & \pi^{\mathsf{T}} \lambda + \lambda' \\
\text{s.t.} \quad & B\pi = \mathbf{1} \qquad Av - b \geq \mathbf{0} \\
& \pi \geq \mathbf{0} \qquad \lambda + \lambda' \mathbf{1} \geq Av - b \\
& \qquad\qquad \lambda, \lambda' \geq \mathbf{0} \\
& \qquad\qquad v \in \mathcal{M}
\end{aligned} \tag{ABP–L_∞}$$

To formulate the mixed integer linear program, assume that there is $\tau \geq \lambda^*$ for the optimal solution λ^*. The mixed integer linear program formulation that corresponds to (ABP–L_∞) is:

$$\begin{aligned}
\min_{z,\pi,\lambda,\lambda',v} \quad & \mathbf{1}^{\mathsf{T}} z + \lambda' \\
\text{s.t.} \quad & z \geq \lambda - \tau(\mathbf{1} - \pi) \\
& B\pi = \mathbf{1} \\
& \lambda + \lambda' \mathbf{1} \geq Av - b \\
& Av - b \geq \mathbf{0} \\
& \lambda, \lambda' \geq \mathbf{0} \\
& v \in \mathcal{M} \\
& \pi \in \{0,1\}^n
\end{aligned} \tag{ABP-MILP}$$

For an appropriate n.

The following theorem states the correctness of this formulation:

Theorem 7.1. *Let $(\pi_1, \lambda_1, \lambda_1')$ be an optimal (greedy-policy) solution of* (ABP–L_∞)*. Then:*

$$\left(\pi_1, \lambda_1, \lambda_1', z' = \min_{z \geq \lambda_1 - (\tau - \pi_1)} \mathbf{1}^{\mathsf{T}} z\right)$$

is an optimal solution of (ABP-MILP) *and vice versa, given that $\tau \geq \lambda_1$. When in addition f_1 and f_2 are the optimal objective values of* (ABP–L_∞) *and* (ABP-MILP)*, then $f_1 = f_2$.*

The proof of the theorem can be found in Section C.8.

This mixed integer linear program formulation is much simpler than the general version. It is much smaller and easier to analyze. Finally, it preserves the structure of the bilinear program and can be used to derive insights into the bilinear formulation.

The actual performance of these solution methods strongly depends on the actual structure of the problem. As usual in solving NP hard problems, there is very little understanding of the theoretical properties that could guarantee faster solution methods. Our informal experiments, however, show that the ABP-specific formulation can solve problems that are orders of magnitude larger than problems that can be solved by the general formulation.

7.4 Homotopy Methods

In this section, we analyze the properties of a homotopy method for solving regularized bilinear programs. While we do not propose a specific method, we focus on the properties of the optimal objective value as a function of ψ.

Approximate bilinear programs can be, just like approximate linear programs, formulated in terms of a regularization coefficient ψ. For the sake of clarity, we only focus on the robust formulation (ABP–L_∞). The set of representable value functions is in that case:

$$\mathcal{M} = \{\Phi x \mid \|x\|_{1,e} \le \psi\}.$$

We consider only L_1 regularization for the representable value functions. The main advantage of the L_1 norm in our setting is that it can be formulated using linear constraints. Many of the results in this section also apply to regularization using other norms. The regularized formulation of the approximate bilinear program is:

$$\begin{aligned}
\min_{\pi \mid \lambda, \lambda', x} \quad & \pi^{\mathsf{T}} \lambda + \lambda' \\
\text{s.t.} \quad & B\pi = \mathbf{1} \qquad A\Phi x - b \ge \mathbf{0} \\
& \pi \ge \mathbf{0} \qquad \lambda + \lambda' \mathbf{1} \ge A\Phi x - b \\
& \qquad\qquad \lambda, \lambda' \ge \mathbf{0} \\
& \qquad\qquad \|x\|_{1,e} \le \psi
\end{aligned} \tag{7.3}$$

First, we analyze the properties of the formulation. An advantageous property of L_1 regularized approximate linear programs, is that their optimal objective value is a convex function of the regularization coefficient ψ. Convexity is an important property in automatically calculating ψ, as described in Section 10.3. Approximate bilinear programs, unlike approximate linear programs, are not convex optimization problems. We now analyze the optimal solution of (7.3) as a function of ψ.

The remainder of the section shows that the optimal objective function of (7.3) is not a convex function of ψ. However, we show that additional assumptions on the structure of the problem enable its use in the feature selection, as proposed in Section 10.3.

Definition 7.2. The optimal solution of (7.3) as a function of ψ is denoted as $x(\psi)$, $\lambda(\psi)$, $\lambda'(\psi)$, $\pi(\psi)$, assuming that the solution is a singleton. This is the optimal solution, not the optimal objective value. The optimal objective value of (7.3) is denoted as $\theta_B(\psi)$:

$$\theta_B(\psi) = \min_{v \in \mathcal{M}(\psi) \cap \mathcal{K}} \|v - Lv\|_\infty.$$

The following proposition states that the function $\theta_B(\psi)$ is not a convex function of ψ.

Proposition 7.3. *The function $\theta_B(\psi)$ is not necessarily convex or concave.*

The proof of the proposition can be found in Section C.8. The intuitive reason for the non-convexity of θ_B is that it is a minimum over a set of convex functions. Clearly, for a fixed policy, the function is convex, since ABP reduces to a linear program for a fixed policy π. The ABP formulation minimizes the Bellman residual over the set of policies; and a minimum of a set of convex functions may not be convex.

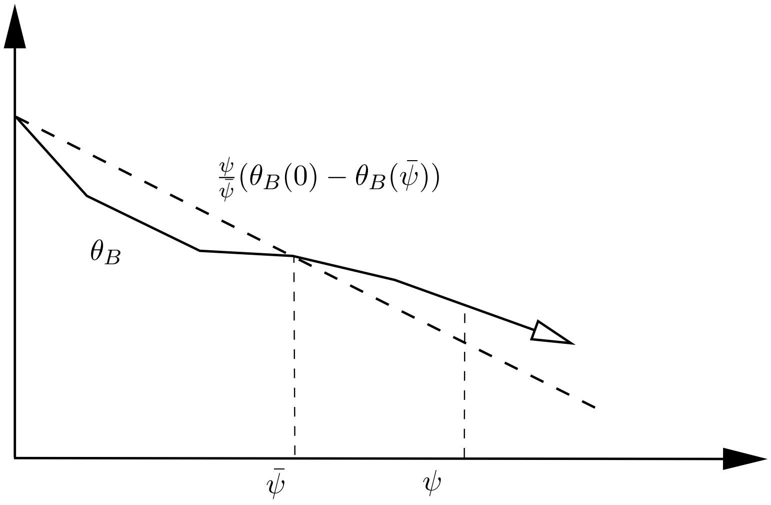
$\frac{\psi}{\bar{\psi}}(\theta_B(0) - \theta_B(\bar{\psi}))$
θ_B
$\bar{\psi}$
ψ

Figure 7.1. An illustration of θ_B and the property satisfied by $\theta_B(\bar{\psi})$.

The importance of the convexity of θ_B is in finding the optimal value ψ using the homotopy method. When the function is convex, there is only a single minimum and the homotopy method may terminate once the function θ_B does stops decreasing. This is not possible when the function is non-convex.

While the function θ_B may not be always convex it still possible to determine $\bar{\psi}$ such that the minimum of the function θ_B is in the interval $[0, \bar{\psi}]$. The following theorem states this property when the rewards for all actions in every state are the same.

Theorem 7.4. *Assume that for every state $s \in \mathcal{S}$ we have that $r(s, a_1) = r(s, a_2)$ for all $a_1, a_2 \in \mathcal{A}$. Also assume that $e(1) = 0$ and $e(i) > 0$ for all $i > 0$. For any $\bar{\psi}$ and $\psi \geq \bar{\psi}$ the function θ_B satisfies that:*

$$\theta_B(\psi) \geq \theta_B(0) - \frac{\psi}{\bar{\psi}}(\theta_B(0) - \theta_B(\bar{\psi})).$$

The proof of the theorem can be found in Section C.8. The theorem states that the function θ_B is very close to being convex. In particular, it satisfied the convexity assumption for point $\psi_1 = 0$ and arbitrary ψ_2. Figure 7.1 illustrates this property of θ_B.

Notice that any MDP can be transformed to have rewards independent of actions by introducing additional auxiliary states. These additional states may, however, influence by the set of representable value functions.

The property in Theorem 7.4 satisfied for MDPs that do have different rewards for actions in the same state. It requires that when $\psi = 0$, the Bellman residual is the same for all actions of a state (the Bellman residual for an action is: $v - L_a v$). Then, when taking a convex combination of the initial value function with another value function, the action with the minimal Bellman residual is the same for the whole set of convex combinations.

There are a variety of possible homotopy algorithms that could be developed for bilinear programs. These can be based on linear program relaxations of the bilinear programs. One option extend the linear program homotopy method algorithm 6.2 to algorithm 5.1. This entails tracing the optimal solution of the linear, and switching the policy to a greedy one during the computation. This method does not guarantee the optimality, however. The best homotopy method for solving bilinear program will need to be based on a successful method for solving bilinear programs and can be expected to be a small extension of the linear program homotopy method. It is first necessary to determine the appropriate method for solving bilinear programs directly.

7.5 Contributions

Most of the derivations in this chapter are new, though they often rely on standard techniques. The derivation of the general mixed integer linear formulation uses some specific properties of the approximate bilinear formulation. The main contributions are the mixed integer linear formulation, described in Section 7.3 and the properties with respect to ψ, which are described in Section 7.4. While we do not present any specific homotopy method for solving bilinear programs, these could be derived from the homotopy method for solving linear programs.

CHAPTER 8

SOLVING SMALL-DIMENSIONAL BILINEAR PROGRAMS

Approximate bilinear formulations are inherently hard to solve as the NP hardness results show. Chapter 7 presents several approaches for solving bilinear programs, which may work very well in some circumstances, but not always. The complexity of solving a bilinear program is due to the size of the bilinear terms. In this chapter, we consider simplifying bilinear programs by explicitly restricting the number of variables involved in the bilinear formulation.

Not all approximate bilinear programs can be formulated to have a small number of bilinear terms. The results in this chapter only apply to the bilinear program (ABP–U). This formulation minimizes a weighted L_1 on the Bellman residual. The benefit of this approach is not only simpler bilinear programs to be solved, but also a smaller risk of overfitting.

This chapter presents an anytime approximate algorithms for solving bilinear programs with a small dimensionality of the bilinear term. First, Section 8.1 defines the bilinear programs, as we use them in this chapter. Then Section 8.2 shows how to reduce the number of bilinear terms — called dimensionality — in a general program and how this applies to approximate bilinear programs. The basic version of the actual approximate algorithm — successive approximation algorithm — is presented in Section 8.3. Section 8.4 shows how to calculate the approximation error of the bilinear algorithm and Section 8.5 shows how to sharpen the bounds and improve the algorithm. Finally, Section 8.6 shows an offline error bound that can provide worst-case guarantees.

8.1 Bilinear Program Formulations

This chapter departs from the notation of bilinear programs and considers bilinear programs as maximization. This is due to historical reasons and to maintain consistency with previous work (Petrik & Zilberstein, 2009).

Definition 8.1. A *separable* bilinear program in the normal form is defined as follows:

$$\begin{aligned}\max_{w,x,y,z} \quad & f(w,x,y,z) = s_1^\mathsf{T} w + r_1^\mathsf{T} x + x^\mathsf{T} C y + r_2^\mathsf{T} y + s_2^\mathsf{T} z \\ \text{s.t.} \quad & A_1 x + B_1 w = b_1 \\ & A_2 y + B_2 z = b_2 \\ & w, x, y, z \geq \mathbf{0}\end{aligned} \tag{8.1}$$

The *size* of the program is the total number of variables in w, x, y and z. The number of variables in y determines the *dimensionality* of the program[1].

[1] It is possible to define the dimensionality in terms of x, or the minimum of dimensions of x and y.

Unless otherwise specified, all vectors are column vectors. We use boldface $\mathbf{0}$ and $\mathbf{1}$ to denote vectors of zeros and ones respectively of the appropriate dimensions. This program specifies two linear programs that are connected only through the nonlinear objective function term $x^{\mathsf{T}}Cy$. The program contains two types of variables. The first type includes the variables x, y that appear in the bilinear term of the objective function. The second type includes the additional variables w, z that do not appear in the bilinear term. As we show later, this distinction is important because the complexity of the algorithm we propose depends mostly on the dimensionality of the problem, which is the number of variables y involved in the bilinear term.

The bilinear program in (8.1) is *separable* because the constraints on x and w are independent of the constraints on y and z. That is, the variables that participate in the bilinear term of the objective function are *independently* constrained. The theory of non-separable bilinear programs is much more complicated and the corresponding algorithms are not as efficient (Horst & Tuy, 1996). Thus, we limit the discussion in this work to separable bilinear programs and often omit the term "separable". As discussed later in more detail, a *separable* bilinear program may be seen as a concave minimization problem with multiple local minima. It can be shown that solving this problem is NP-complete, compared to polynomial time complexity of linear programs.

In addition to the formulation of the bilinear program shown in (8.1), we also use the following formulation, stated in terms of inequalities:

$$\begin{aligned} \max_{x,y} \quad & x^{\mathsf{T}}Cy \\ \text{s.t.} \quad & A_1x \leq b_1 \quad x \geq \mathbf{0} \\ & A_2y \leq b_2 \quad y \geq \mathbf{0} \end{aligned} \tag{8.2}$$

The latter formulation can be easily transformed into the normal form using standard transformations of linear programs (Vanderbei, 2001). In particular, we can introduce slack variables w, z to obtain the following identical bilinear program in the normal form:

$$\begin{aligned} \max_{w,x,y,z} \quad & x^{\mathsf{T}}Cy \\ \text{s.t.} \quad & A_1x - w = b_1 \\ & A_2y - z = b_2 \\ & w, x, y, z \geq \mathbf{0} \end{aligned} \tag{8.3}$$

As we show later, the presence of the variables w, z in the objective function may prevent a crucial function from being convex. Since this has an unfavorable impact on the properties of the bilinear program, we introduce a compact form of the problem.

Definition 8.2. We say that the bilinear program in (8.1) is in a *compact* form when s_1 and s_2 are zero vectors. It is in a *semi-compact* form if s_2 is a zero vector.

The compactness requirement is not limiting because any bilinear program in the form shown in (8.1) can be expressed in a semi-compact form as follows:

$$\begin{aligned} \max_{w,x,y,z,\hat{x},\hat{y}} \quad & s_1^{\mathsf{T}} w + r_1^{\mathsf{T}} x + \begin{pmatrix} x^{\mathsf{T}} & \hat{x} \end{pmatrix} \begin{pmatrix} C & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y \\ \hat{y} \end{pmatrix} + r_2^{\mathsf{T}} y \\ \text{s.t.} \quad & A_1 x + B_1 w = b_1 \quad A_2 y + B_2 z = b_2 \\ & \hat{x} = 1 \quad \hat{y} = s_2^{\mathsf{T}} z \\ & w, x, y, z \geq \mathbf{0} \end{aligned} \tag{8.4}$$

Clearly, feasible solutions of (8.1) and (8.4) have the same objective value when $\hat{y}$ is set appropriately. Notice that the dimensionality of the bilinear term in the objective function increases by 1 for both x and y. Hence, this transformation increases the dimensionality of the program by 1.

8.2 Dimensionality Reduction

In this section, we show the principles behind automatically determining the necessary dimensionality of a given problem. This the procedure applies to bilinear programs that minimize L_1 norm, but also works for general bilinear programs.

The dimensionality is inherently part of the model, not the problem itself. There may be equivalent models of a given problem with very different dimensionality. Thus, procedures for reducing the dimensionality are not necessary when the modeler can create a model with minimal dimensionality. However, this is nontrivial in many cases. In addition, some dimensions may have little impact on the overall performance. To determine which ones can be discarded, we need a measure of their contribution that can be computed efficiently. We define these notions more formally later in this section.

We assume that the feasible sets have bounded L_2 norms, and assume a general formulation of the bilinear program, not necessarily in the semi-compact form. Given Assumption 8.7, this can be achieved by scaling the constraints when the feasible region is bounded.

Assumption 8.3. For all $x \in X$ and $y \in Y$, their norms satisfy $\|x\|_2 \leq 1$ and $\|y\|_2 \leq 1$.

We discuss the implications of and problems with this assumption after presenting Theorem 8.4. Intuitively, the dimensionality reduction removes those dimensions where $g(y)$ is constant, or almost constant. Interestingly, these dimensions may be recovered based on the eigenvectors and eigenvalues of $C^{\mathsf{T}}C$. We use the eigenvectors of $C^{\mathsf{T}}C$ instead of the eigenvectors of C, because our analysis is based on L_2 norm of x and y and thus of C. The L_2 norm $\|C\|_2$ is bounded by the largest eigenvalue of $C^{\mathsf{T}}C$. In addition, a symmetric matrix is required to ensure that the eigenvectors are perpendicular and span the whole space.

Given a problem represented using (8.1), let F be a matrix whose columns are all the eigenvectors of $C^{\mathsf{T}}C$ with eigenvalues greater than some $\bar{\lambda}$. Let G be a matrix with all the remaining eigenvectors as columns. Notice that together, the columns of the matrices span the whole space and are real-valued, since $C^{\mathsf{T}}C$ is a symmetric matrix. Assume without

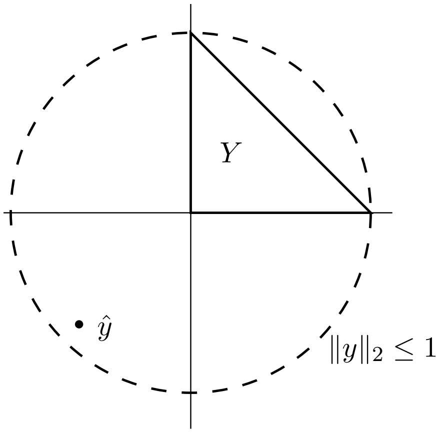

Figure 8.1. Approximation of the feasible set Y according to Assumption 8.3.

loss of generality that the eigenvectors are unitary. The *compressed version* of the bilinear program is then the following:

$$
\begin{aligned}
\max_{w,x,y_1,y_2,z} \quad & \tilde{f}(w,x,y_1,y_2,z) = r_1^\mathsf{T} x + s_2^\mathsf{T} w + x^\mathsf{T} C F y_1 + r_2^\mathsf{T} \begin{pmatrix} F & G \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} + s_2^\mathsf{T} z \\
\text{s.t.} \quad & A_1 x + B_1 w = b \\
& A_2 \begin{pmatrix} F & G \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} + B_2 z = b_2 \\
& w, x, y_1, y_2, z \geq \mathbf{0}
\end{aligned}
\tag{8.5}
$$

Notice that the program is missing the element $x^\mathsf{T} C G y_2$, which would make its optimal solutions identical to the optimal solutions of (8.1). A more practical approach to reducing the dimensionality would be based on singular vector decomposition. This approach is based on singular value decomposition and may be directly applied to any bilinear program. The following theorem quantifies the maximum error when using the compressed program.

Theorem 8.4. *Let f^* and $\tilde{f}^*$ be optimal solutions of* (8.1) *and* (8.5) *respectively. Then:*

$$\epsilon = |f^* - \tilde{f}^*| \leq \sqrt{\lambda}.$$

Moreover, this is the maximal linear dimensionality reduction possible with this error without considering the constraint structure.

The proof of the theorem can be found in Section C.9.

Alternatively, the bound can be proved by replacing the equality $A_1 x + B_1 w = b_1$ by $\|x\|_2 = 1$. The bound can then be obtained by Lagrange necessary optimality conditions. In these bounds we use L_2-norm; an extension to a different norm is not straightforward. Note also that this dimensionality reduction technique ignores the constraint structure. When the constraints have some special structure, it might be possible to obtain an even tighter bound. As described in the next section, the dimensionality reduction technique generalizes the reduction that Becker, Zilberstein, Lesser, and Goldman (2004) used *implicitly.*

The result of Theorem 8.4 is based on an approximation of the feasible set Y by $\|y\|_2 \leq 1$, as Assumption 8.3 states. This approximation may be quite loose in some problems, which may lead to a significant *multiplicative* overestimation of the bound in Theorem 8.4. For example, consider the feasible set depicted in Figure 8.1. The bound may be achieved in a point $\hat{y}$, which is far from the feasible region. In specific problems, a tighter bound could be obtained by either appropriately scaling the constraints, or using a weighted L_2 with a better precision. We partially address this issue by considering the structure of the constraints. To derive this, consider the following linear program and corresponding theorem:

$$\begin{aligned} \max_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax = b \quad x \geq \mathbf{0} \end{aligned} \tag{8.6}$$

Theorem 8.5. *The optimal solution of* (8.6) *is the same as when the objective function is modified to*

$$c^{\mathsf{T}}(\mathbf{I} - A^{\mathsf{T}} \left(AA^{\mathsf{T}}\right)^{-1} A)x.$$

Proof. The objective function is:

$$\begin{aligned} \max_{\{x \,|\, Ax=b,\, x\geq \mathbf{0}\}} c^{\mathsf{T}} x \quad &= \\ &= \max_{\{x \,|\, Ax=b,\, x\geq \mathbf{0}\}} c^{\mathsf{T}}(\mathbf{I} - A^{\mathsf{T}} \left(AA^{\mathsf{T}}\right)^{-1} A)x + c^{\mathsf{T}} A^{\mathsf{T}} \left(AA^{\mathsf{T}}\right)^{-1} Ax \\ &= c^{\mathsf{T}} A^{\mathsf{T}} \left(AA^{\mathsf{T}}\right)^{-1} b + \max_{\{x \,|\, Ax=b,\, x\geq \mathbf{0}\}} c^{\mathsf{T}}(\mathbf{I} - A^{\mathsf{T}} \left(AA^{\mathsf{T}}\right)^{-1} A)x. \end{aligned}$$

The first term may be ignored because it does not depend on the solution x. □

The following corollary shows how the above theorem can be used to strengthen the dimensionality reduction bound. For example, in zero-sum games, this stronger dimensionality reduction splits the bilinear program into two linear programs.

Corollary 8.6. *Assume that there are no variables w and z in* (8.1)*. Let:*

$$\mathcal{Q}_i = (\mathbf{I} - A_i^{\mathsf{T}} \left(A_i A_i^{\mathsf{T}}\right)^{-1} A_i)), \quad i \in \{1, 2\},$$

where A_i are defined in (8.1)*. Let $\tilde{C}$ be:*

$$\tilde{C} = \mathcal{Q}_1 C \mathcal{Q}_2,$$

where C is the bilinear-term matrix from (8.1)*. Then the bilinear programs will have identical optimal solutions with either C or $\tilde{C}$.*

Proof. Using Theorem 8.5, we can modify the original objective function in (8.1) to:

$$f(x, y) = r_1^{\mathsf{T}} x + x^{\mathsf{T}}(\mathbf{I} - A_1^{\mathsf{T}} \left(A_1 A_1^{\mathsf{T}}\right)^{-1} A_1))C(\mathbf{I} - A_2^{\mathsf{T}} \left(A_2 A_2^{\mathsf{T}}\right)^{-1} A_2))y + r_2^{\mathsf{T}} y.$$

For the sake of simplicity we ignore the variables w and z, which do not influence the bilinear term. Because both $(\mathbf{I} - A_i^{\mathsf{T}} \left(A_i A_i^{\mathsf{T}}\right)^{-1} A_i)$ for $i = 1, 2$ are orthogonal projection matrices, none of the eigenvalues in Theorem 8.4 will increase. □

The bilinear program that minimizes the expected policy loss:

$$\begin{aligned} \min_{\pi \mid \lambda, v} \quad & \pi^{\mathsf{T}} U \lambda - \alpha^{\mathsf{T}} v & & \\ \text{s.t.} \quad & B\pi = \mathbf{1} & & Av - b \geq \mathbf{0} \\ & \pi \geq \mathbf{0} & & \lambda = Av - b \\ & & & v \in \mathcal{M} \end{aligned} \tag{ABP–U}$$

Although this bilinear program represents a minimization, this is immaterial to the dimensionality reduction procedure. The formulation above uses the approximation only conceptually by requiring that $v, \lambda \in \mathcal{M}$; to reduce the dimensionality the representation must be used explicitly as:

$$\begin{aligned} \min_{\pi \mid \lambda, x} \quad & \pi^{\mathsf{T}} U (A\Phi x - b) - \alpha^{\mathsf{T}} \Phi x & & \\ \text{s.t.} \quad & B\pi = \mathbf{1} & & A\Phi x - b \geq \mathbf{0} \\ & \pi \geq \mathbf{0} & & \|x\|_2 \leq \psi \end{aligned} \tag{8.7}$$

Here, $v = \Phi x$. The matrix $C = UA\Phi$ has at most m non-zero eigenvectors — where m is the number of features — because its rank is at most m. Using Theorem 8.4, there exists an identical bilinear program with dimensionality m.

The dimensionality reduction proposed in this chapter requires that the feasible regions are bounded. This is the main reason why the bilinear program in (8.7) includes regularization constraints. While the dimensionality reduction in this chapter does require that the feasible sets have a bounded L_2 norm, this is only necessary for excluding dimensions with non-zero eigenvalue. It is also possible to solve the approximate bilinear program with a large number of features, when the regularization can be used to eliminate dimensions with non-zero eigenvalues. The algorithm for solving bilinear programs described in this chapter can be practical for ABPs with up to about 50 features.

8.3 Successive Approximation Algorithm

The rest of this section presents a new anytime algorithm for solving bilinear programs. The goal of the algorithm to is to produce a good solution quickly and then improve the solution in the remaining time. Along with each approximate solution, the maximal approximation bound with respect to the optimal solution is provided. As we show below, our algorithm can benefit from results produced by suboptimal algorithms, such as Algorithm 5.1, to quickly determine tight approximation bounds.

We begin with an overview of a successive approximation algorithm for bilinear problems, which takes advantage of a small number of bilinear terms. It is particularly suitable when the input problem is large in comparison to its dimensionalit. We address the issue of dimensionality reduction in Section 8.2.

We begin with a simple intuitive explanation of the algorithm, and then show how it can be formalized. The bilinear program can be seen as an optimization game played by two agents, in which the first agent sets the variables w, x and the second one sets the variables y, z. This is a general observation that applies to any bilinear program. In any practical application, the feasible sets for the two sets of variables may be too large to explore exhaustively. In fact, when this method is applied to DEC-MDPs, these sets are infinite and continuous. The basic idea of the algorithm is to first identify the set of best responses of one of the agents, say agent 1, to some policy of the other agent. This is simple because once the variables of agent 2 are fixed, the program becomes linear, which is relatively easy to solve. Once the set of best-response policies of agent 1 is identified, assuming it is of a reasonable size, it is possible to calculate the best response of agent 2.

This general approach is also used by the coverage set algorithm (Becker et al., 2004). One distinction is that the representation used in CSA applies only to DEC-MDPs, while our formulation applies to bilinear programs–a more general representation. The main distinction between our algorithm and CSA is the way in which the variables y, z are chosen. In CSA, the values y, z are calculated in a way that simply guarantees termination in finite time. We, on the other hand, choose values y, z greedily so as to minimize the approximation bound on the optimal solution. This is possible because we establish bounds on the optimality of the solution throughout the calculation. As a result, our algorithm converges more rapidly and may be terminated at any time with a guaranteed performance bound. Unlike the earlier version of the algorithm (Petrik & Zilberstein, 2007a), the version described here calculates the best response using only a subset of the values of y, z. As we show, it is possible to identify regions of y, z in which it is impossible to improve the current best solution and exclude these regions from consideration.

We now formalize the ideas described above. To simplify the notation, we define feasible sets as follows:

$$\begin{aligned} X &= \{(x, w) \mid A_1 x + B_1 w = b_1\} \\ Y &= \{(y, z) \mid A_2 y + B_2 z = b_2\}. \end{aligned}$$

We use $y \in Y$ to denote that there exists z such that $(y, z) \in Y$. In addition, we assume that the problem is in a semi-compact form. This is reasonable because any bilinear program may be converted to semi-compact form with an increase in dimensionality of one, as we have shown earlier.

Assumption 8.7. The sets X and Y are bounded, that is, they are contained in a ball of a finite radius.

While Assumption 8.7 is limiting, coordination problems under uncertainty typically have bounded feasible sets because the variables correspond to probabilities bounded to $[0, 1]$.

Assumption 8.8. The bilinear program is in a semi-compact form.

The main idea of the algorithm is to compute a set $\tilde{X} \subseteq X$ that contains only those elements that satisfy a necessary optimality condition. The set $\tilde{X}$ is formally defined as follows:

$$\tilde{X} \subseteq \left\{ (x^*, w^*) \;\middle|\; \exists (y, z) \in Y \; f(w^*, x^*, y, z) = \max_{(x,w) \in X} f(w, x, y, z) \right\}.$$

As described above, this set may be seen as a set of best responses of one agent to the variable settings of the other. The best responses are easy to calculate since the bilinear program in (8.1) reduces to a linear program for fixed w, x or fixed y, z. In our algorithm, we assume that $\tilde{X}$ is potentially a proper subset of all necessary optimality points and focus on the approximation error of the optimal solution. Given the set $\tilde{X}$, the following simplified problem is solved.

$$\begin{aligned} \max_{w,x,y,z} \quad & f(w,x,y,z) \\ \text{s.t.} \quad & (x,w) \in \tilde{X} \\ & A_2 y + B_2 z = b_2 \\ & y, z \geq \mathbf{0} \end{aligned} \tag{8.8}$$

Unlike the original continuous set X, the reduced set $\tilde{X}$ is discrete and small. Thus the elements of $\tilde{X}$ may be enumerated. For a fixed w and x, the bilinear program in (8.8) reduces to a linear program.

To help compute the approximation bound and to guide the selection of elements for $\tilde{X}$, we use the *best-response function* $g(y)$, defined as follows:

$$g(y) = \max_{\{w,x,z \,|\, (x,w) \in X, (y,z) \in Y\}} f(w,x,y,z) = \max_{\{x,w \,|\, (x,w) \in X\}} f(w,x,y,0),$$

with the second equality for semi-compact programs only and feasible $y \in Y$. Note that $g(y)$ is also defined for $y \notin Y$, in which case the choice of z is arbitrary since it does not influence the objective function. The best-response function is easy to calculate using a linear program. The crucial property of the function g that we use to calculate the approximation bound is its *convexity*. The following proposition holds because $g(y) = \max_{\{x,w \,|\, (x,w) \in X\}} f(w,x,y,0)$ is a maximum of a finite set of linear functions.

Proposition 8.9. *The function $g(y)$ is convex when the program is in a semi-compact form.*

Proposition 8.9 relies heavily on the separability of (8.1), which means that the constraints on the variables on one side of the bilinear term are independent of the variables on the other side. The separability ensures that w, x are valid solutions regardless of the values of y, z. The semi-compactness of the program is necessary to establish convexity, as shown in C.25 in Section C.9.1. The example is constructed using the properties described in the appendix, which show that $f(w,x,y,z)$ may be expressed as a sum of a convex and a concave function.

We are now ready to describe Algorithm 8.1, which computes the set $\tilde{X}$ for a bilinear problem $\mathcal{B}$ such that the approximation error is at most ϵ_0. The algorithm iteratively adds the best response (x,w) for a selected *pivot point* y into $\tilde{X}$. The pivot points are selected hierarchically. At an iteration j, the algorithm keeps a set of polyhedra $S_1 \ldots S_j$ which represent the triangulation of the feasible space Y, which is possible based on Assumption 8.7. For each polyhedron $S_i = (y_1 \ldots y_{n+1})$, the algorithm keeps a bound ϵ_i on the maximal difference between the optimal solution on the polyhedron and the best solution found so far. This error bound on a polyhedron S_i is defined as:

$$\epsilon_i = e(S_i) = \max_{\{w,x,y | (x,w) \in X, y \in S_i\}} f(w,x,y,0) - \max_{\{w,x,y | (x,w) \in \tilde{X}, y \in S_i\}} f(w,x,y,0),$$

where $\tilde{X}$ represents the current, not final, set of best responses.

Next, a point y_0 is selected as described below and $n+1$ new polyhedra are created by replacing one of the vertices by y_0 to get: $(y_0, y_2, \ldots), (y_1, y_0, y_3, \ldots), \ldots, (y_1, \ldots, y_n, y_0)$. This is depicted for a 2-dimensional set Y in Figure 8.2. The old polyhedron is discarded

Algorithm 8.1: BestResponseApprox($\mathcal{B}, \epsilon_0$) returns (w, x, y, z)

```
// Create the initial polyhedron $S_1$.
$S_1 \leftarrow (y_1 \dots y_{n+1}), Y \subseteq S_1$ ;
// Add best-responses for vertices of $S_1$ to $\tilde{X}$
$\tilde{X} \leftarrow \{\arg\max_{(x,w)\in X} f(w, x, y_1, 0), ..., \arg\max_{(x,w)\in X} f(w, x, y_{n+1}, 0)\}$ ;
// Calculate the error $\epsilon$ and pivot point $\phi$ of the initial polyhedron
$(\epsilon_1, \phi_1) \leftarrow PolyhedronError(S_1)$ ;                    // Section 8.4,Section 8.5
// Initialize the number of polyhedra to 1
$j \leftarrow 1$ ;
// Continue until reaching a predefined precision $\epsilon_0$
while $\max_{i=1,\dots,j} \epsilon_i \geq \epsilon_0$ do
    // Find the polyhedron with the largest error
    $i \leftarrow \arg\max_{k=1,\dots,j} \epsilon_k$ ;
    // Select the pivot point of the polyhedron with the largest error
    $y \leftarrow \phi_i$ ;
    // Add the best response to the pivot point $y$ to the set $\tilde{X}$
    $\tilde{X} \leftarrow X \cup \{\arg\max_{(x,w)\in X} f(w, x, y, 0)\}$ ;
    // Calculate errors and pivot points of the refined polyhedra
    for $k = 1, \dots, n+1$ do
        $j \leftarrow j + 1$ ;
        // Replace the $k$-th vertex by the pivot point $y$
        $S_j \leftarrow (y, y_1 \dots y_{k-1}, y_{k+1}, \dots y_{n+1})$ ;
        $(\epsilon_j, \phi_j) \leftarrow PolyhedronError(S_j)$ ;          // Section 8.4,Section 8.5
        // Take the smaller of the errors on the original and the refined
           polyhedron.  The error may not increase with the refinement,
           although the bound may
        $\epsilon_j \leftarrow \min\{\epsilon_i, \epsilon_j\}$ ;
    // Set the error of the refined polyhedron to 0, since the region is
       covered by the refinements
    $\epsilon_i \leftarrow 0$ ;
$(w, x, y, z) \leftarrow \arg\max_{\{w,x,y,z \,|\, (x,w)\in\tilde{X}, (y,z)\in Y\}} f(w, x, y, 0)$ ;
return $(w, x, y, z)$ ;
```

and the above procedure is then repeatedly applied to the polyhedron with the maximal approximation error.

For the sake of clarity, the pseudo-code of Algorithm 8.1 is simplified and does not address any efficiency issues. In practice, $g(y_i)$ could be cached, and the errors ϵ_i could be stored in a prioritized heap or at least in a sorted array. In addition, a lower bound l_i and an upper bound u_i is calculated and stored for each polyhedron $S_i = (y_1 \dots y_{n+1})$. The function $e(S_i)$ calculates their maximal difference on the polyhedron S_i and the point where it is attained. The error bound ϵ_i on the polyhedron S_i may not be tight, as we describe in Remark 8.13. As a result, when the polyhedron S_i is refined to n polyhedra $S'_1 \dots S'_n$ with online error bounds $\epsilon'_1 \dots \epsilon'_n$, it is possible that for some k: $\epsilon'_k > \epsilon_i$. Since $S'_1 \dots S'_n \subseteq S_i$, the true error on S'_k is less than on S_i and therefore ϵ'_k may be set to ϵ_i.

Conceptually, the algorithm is similar to CSA, but there are some important differences. The main difference is in the choice of the pivot point y_0 and the bounds on g. CSA does not keep any upper bound and it evaluates $g(y)$ on all the intersection points of planes defined by the current solutions in $\tilde{X}$. That guarantees that $g(y)$ is eventually known

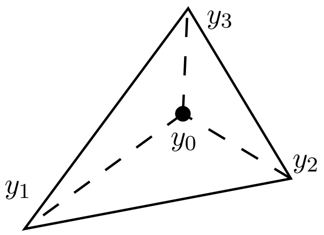

Figure 8.2. Refinement of a polyhedron in two dimensions with a pivot y_0.

precisely (Becker et al., 2004). A similar approach was also taken for POMDPs (Cheng, 1988). The upper bound on the number of intersection points in CSA is $\binom{|\tilde{X}|}{\dim Y}$. The principal problem is that the bound is exponential in the dimension of Y, and experiments do not show a slower growth in typical problems. In contrast, we choose the pivot points to minimize the approximation error. This is more selective and tends to more rapidly reduce the error bound. In addition, the error at the pivot point may be used to determine the overall error bound. The following proposition states the soundness of the triangulation. The correctness of the triangulation establishes that in each iteration the approximation error over Y is equivalent to the maximum of the approximation errors over the current polyhedra $S_1 \dots S_j$.

Proposition 8.10. *In the proposed triangulation, the sub-polyhedra do not overlap and they cover the whole feasible set Y, given that the pivot point is in the interior of S.*

The proof of the proposition can be found in Section C.9.

8.4 Online Error Bound

The selection of the pivot point plays a key role in the performance of the algorithm, in both calculating the error bound and the speed of convergence to the optimal solution. In this section we show exactly how we use the triangulation in the algorithm to calculate an error bound. To compute the approximation bound, we define the *approximate best-response function* $\tilde{g}(y)$ as:

$$\tilde{g}(y) = \max_{\{x,w \,|\, (x,w)\in\tilde{X}\}} f(w,x,y,0).$$

Notice that z is not considered in this expression, since we assume that the bilinear program is in the semi-compact form. The value of the best approximate solution during the execution of the algorithm is:

$$\max_{\{w,x,y,z \,|\, (x,w)\in\tilde{X}, y\in Y\}} f(w,x,y,0) = \max_{y\in Y} \tilde{g}(y).$$

This value can be calculated at runtime when each new element of $\tilde{X}$ is added. Then the maximal approximation error between the current solution and the optimal one may be calculated from the approximation error of the best-response function $g(\cdot)$, as stated by the following proposition.

Proposition 8.11. *Consider a bilinear program in a semi-compact form. Then let $\tilde{w}, \tilde{x}, \tilde{y}$ be an optimal solution of* (8.8) *and let w^*, x^*, y^* be an optimal solution of* (8.1). *The approximation error is then bounded by:*

$$f(w^*, x^*, y^*, 0) - f(\tilde{w}, \tilde{x}, \tilde{y}, 0) \leq \max_{y \in Y} \left(g(y) - \tilde{g}(y)\right).$$

Proof.

$$f(w^*, x^*, y^*, 0) - f(\tilde{w}, \tilde{x}, \tilde{y}, 0) = \max_{y \in Y} g(y) - \max_{y \in Y} \tilde{g}(y) \leq \max_{y \in Y} g(y) - \tilde{g}(y)$$

□

Now, the approximation error is $\max_{y \in Y} g(y) - \tilde{g}(y)$, which is bounded by the difference between an upper bound and a lower bound on $g(y)$. Clearly, $\tilde{g}(y)$ is a lower bound on $g(y)$. Given points in which $\tilde{g}(y)$ is the same as the best-response function $g(y)$, we can use Jensen's inequality to obtain the upper bound. This is summarized by the following lemma.

Lemma 8.12. *Let $y_i \in Y$ for $i = 1, \ldots, n+1$ such that $\tilde{g}(y_i) = g(y_i)$. Then $g\left(\sum_{i=1}^{n+1} c_i y_i\right) \leq \sum_{i=1}^{n+1} c_i g(y_i)$ when $\sum_{i=1}^{n+1} c_i = 1$ and $c_i \geq 0$ for all i.*

The actual implementation of the bound relies on the choice of the pivot points. Next we describe the maximal error calculation on a single polyhedron defined by $S = (y_1 \ldots y_n)$. Let matrix T have y_i as columns, and let $L = \{x_1 \ldots x_{n+1}\}$ be the set of the best responses for its vertices. The matrix T is used to convert any y in absolute coordinates to a relative representation t that is a convex combination of the vertices. This is defined formally as follows:

$$\begin{aligned} y &= Tt = \begin{pmatrix} | & | & \ldots \\ y_1 & y_2 & \ldots \\ | & | & \ldots \end{pmatrix} t \\ 1 &= \mathbf{1}^\mathsf{T} t \\ 0 &\leq t \end{aligned}$$

where the y_i's are column vectors.

We can represent a lower bound $l(y)$ for $\tilde{g}(y)$ and an upper bound $u(y)$ for $g(y)$ as:

$$\begin{aligned} l(y) &= \max_{x \in L} r^\mathsf{T} x + x^\mathsf{T} C y \\ u(y) &= [g(y_1), g(y_2), \ldots]^\mathsf{T} t = [g(y_1), g(y_2), \ldots]^\mathsf{T} \begin{pmatrix} T \\ \mathbf{1}^\mathsf{T} \end{pmatrix}^{-1} \begin{pmatrix} y \\ 1 \end{pmatrix}, \end{aligned}$$

The upper bound correctness follows from Lemma 8.12. Notice that $u(y)$ is a linear function, which enables us to use a linear program to determine the maximal-error point.

Remark 8.13. Notice that we use L instead of $\tilde{X}$ in calculating $l(y)$. Using all of $\tilde{X}$ would lead to a tighter bound, as it is easy to show in three-dimensional examples. However, this also would substantially increase the computational complexity.

Algorithm 8.2: PolyhedronError($\mathcal{B}$, S)

```
$\mathcal{P} \leftarrow$ one of (8.9), or (8.10), or (8.11), or (8.17) ;
$t \leftarrow$ the optimal solution of $\mathcal{P}$ ;
$\epsilon \leftarrow$ the optimal objective value of $\mathcal{P}$ ;
// Coordinates $t$ are relative to the vertices of $S$, convert them to
   absolute values in $Y$
$\phi \leftarrow Tt$ ;
return $(\epsilon, \phi)$ ;
```

Now, the error on a polyhedron S may be expressed as:

$$\begin{aligned} e(S) &\leq \max_{y \in S} u(y) - l(y) = \max_{y \in S} u(y) - \max_{x \in L} r^{\mathsf{T}} x + x^{\mathsf{T}} C y \\ &= \max_{y \in S} \min_{x \in L} u(y) - r^{\mathsf{T}} x - x^{\mathsf{T}} C y. \end{aligned}$$

We also have

$$y \in S \Leftrightarrow \left(y = Tt \wedge t \geq \mathbf{0} \wedge \mathbf{1}^{\mathsf{T}} t = 1 \right).$$

As a result, the point with the maximal error bound may be determined using the following linear program in terms of variables t, ϵ:

$$\begin{aligned} \max_{t,\epsilon} \quad & \epsilon \\ \text{s.t.} \quad & \epsilon \leq u(Tt) - r^{\mathsf{T}} x - x^{\mathsf{T}} C T t \quad \forall x \in L \\ & \mathbf{1}^{\mathsf{T}} t = 1 \quad t \geq \mathbf{0} \end{aligned} \tag{8.9}$$

Here x is *not* a variable. The formulation is correct because all feasible solutions are bounded below the maximal error and any maximal-error solution is feasible.

Proposition 8.14. *The optimal solution of* (8.9) *is equivalent to* $\max_{y \in S} |u(y) - l(y)|$.

We thus select the next pivot point to greedily minimize the error. The maximal difference is actually achieved in points where some of the planes meet, as Becker et al. (2004) have suggested. However, checking these intersections is very similar to running the simplex algorithm. In general, the simplex algorithm is preferable to interior point methods for this program because of its small size (Vanderbei, 2001).

algorithm 8.2 shows a general way to calculate the maximal error and the pivot point on the polyhedron S. This algorithm may use the basic formulation in (8.9), or the more advanced formulations in equations (8.10), (8.11), and (8.17) defined in Section 8.5.

In the following section, we describe a more refined pivot point selection method that can in some cases dramatically improve the performance.

8.5 Advanced Pivot Point Selection

As described above, the pivot points are chosen greedily to both determine the maximal error in each polyhedron and to minimize the approximation error. The basic approach described in Section 8.3 may be refined, because the goal is not to approximate the function

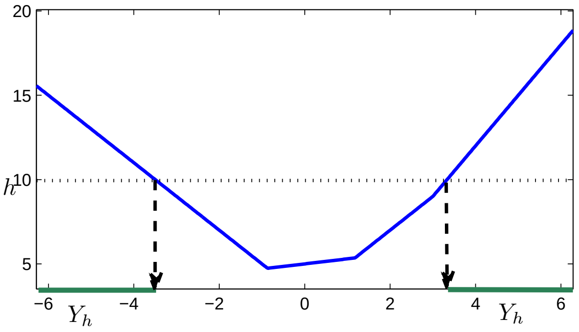

Figure 8.3. The reduced set Y_h that needs to be considered for pivot point selection.

$g(y)$ with the least error, but to find the optimal solution. Intuitively, we can ignore those regions of Y that will not guarantee any improvement of the current solution, as illustrated in Figure 8.3. As we show below, the search for the maximal error point could be limited to this region as well.

We first define a set $Y_h \subseteq Y$ that we will search for the maximal error, given that the optimal solution $f^* \geq h$.

$$Y_h = \{y \,|\, g(y) \geq h, y \in Y\}.$$

The next proposition states that the maximal error needs to be calculated only in a superset of Y_h.

Proposition 8.15. *Let $\tilde{w}, \tilde{x}, \tilde{y}, \tilde{z}$ be the approximate optimal solution and w^*, x^*, y^*, z^* be the optimal solution. Also let $f(w^*, x^*, y^*, z^*) \geq h$ and assume some $\tilde{Y}_h \supseteq Y_h$. The approximation error is then bounded by:*

$$f(w^*, x^*, y^*, z^*) - f(\tilde{w}, \tilde{x}, \tilde{y}, \tilde{z}) \leq \max_{y \in \tilde{Y}_h} g(y) - \tilde{g}(y).$$

Proof. First, $f(w^*, x^*, y^*, z^*) = g(y^*) \geq h$ and thus $y^* \in Y_h$. Then:

$$\begin{aligned} f(w^*, x^*, y^*, z^*) - f(\tilde{w}, \tilde{x}, \tilde{y}, \tilde{z}) &= \max_{y \in Y_h} g(y) - \max_{y \in Y} \tilde{g}(y) \\ &\leq \max_{y \in Y_h} g(y) - \tilde{g}(y) \\ &\leq \max_{y \in \tilde{Y}_h} g(y) - \tilde{g}(y) \end{aligned}$$

□

Proposition 8.15 indicates that the point with the maximal error needs to be selected only from the set Y_h. The question is how to easily identify Y_h. Because the set is not convex in general, a tight approximation of this set needs to be found. In particular, we use methods that approximate the intersection of a superset of Y_h with the polyhedron that is being refined, using the following methods:

1. *Feasibility* [(8.10)]: Require that pivot points are feasible in Y.
2. *Linear bound* [(8.11)]: Use the linear upper bound $u(y) \geq h$.

3. *Cutting plane* [(8.17)]: Use the linear inequalities that define Y_h^{C}, where $Y_h^{\mathsf{C}} = \mathbb{R}^{|Y|} \setminus Y_h$ is the complement of Y_h.

Any combination of these methods is also possible.

Feasibility The first method is the simplest, but also the least constraining. The linear program to find the pivot point with the maximal error bound is as follows:

$$\begin{aligned} \max_{\epsilon,t,y,z} \quad & \epsilon \\ \text{s.t.} \quad & \epsilon \le u(Tt) - r^{\mathsf{T}} x + x^{\mathsf{T}} C T t \quad \forall x \in L \\ & \mathbf{1}^{\mathsf{T}} t = 1 \quad t \ge \mathbf{0} \\ & y = Tt \\ & A_2 y + B_2 z = b_2 \\ & y, z \ge \mathbf{0} \end{aligned} \tag{8.10}$$

This approach does not require that the bilinear program is in the semi-compact form.

Linear Bound The second method, using the linear bound, is also very simple to implement and compute, and it is more selective than just requiring feasibility. Let:

$$\tilde{Y}_h = \{y \,|\, u(y) \ge h\} \supseteq \{y \,|\, g(y) \ge h\} = Y_h.$$

This set is convex and thus does not need to be approximated. The linear program used to find the pivot point with the maximal error bound is as follows:

$$\begin{aligned} \max_{\epsilon,t} \quad & \epsilon \\ \text{s.t.} \quad & \epsilon \le u(Tt) - r^{\mathsf{T}} x + x^{\mathsf{T}} C T t \quad \forall x \in L \\ & \mathbf{1}^{\mathsf{T}} t = 1 \quad t \ge \mathbf{0} \\ & u(Tt) \ge h \end{aligned} \tag{8.11}$$

The difference from (8.9) is the last constraint. This approach requires that the bilinear program is in the semi-compact form to ensure that $u(y)$ is a bound on the total return.

Cutting Plane The third method, using the cutting plane elimination, is the most computationally intensive one, but also the most selective one. Using this approach requires additional assumptions on the other parts of the algorithm, which we discuss below. The method is based on the same principle as α-extensions in concave cuts (Horst & Tuy, 1996). We start with the set Y_h^{C} because it is convex and may be expressed as:

$$\left(\max_{w,x} s_1^{\mathsf{T}} w + r_1^{\mathsf{T}} x + y^{\mathsf{T}} C^{\mathsf{T}} x + r_2^{\mathsf{T}} y \right) \le h \tag{8.12}$$

$$A_1 x + B_1 w = b_1 \tag{8.13}$$

$$w, x \ge \mathbf{0} \tag{8.14}$$

To use these inequalities in selecting the pivot point, we need to make them linear. But there are two obstacles: (8.12) contains a bilinear term and is a maximization. Both of

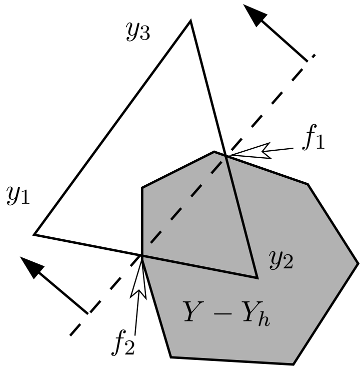

Figure 8.4. Approximating Y_h using the cutting plane elimination method.

these issues can be addressed by using the dual formulation of (8.12). The corresponding linear program and its dual for fixed y, ignoring constants h and $r_2^{\mathsf{T}} y$, are:

$$\begin{aligned} \max_{w,x} \quad & s_1^{\mathsf{T}} w + r_1^{\mathsf{T}} x + y^{\mathsf{T}} C^{\mathsf{T}} x \\ \text{s.t.} \quad & A_1 x + B_1 w = b_1 \\ & w, x \geq \mathbf{0} \end{aligned} \tag{8.15}$$

and

$$\begin{aligned} \min_{\lambda} \quad & b_1^{\mathsf{T}} \lambda \\ \text{s.t.} \quad & A_1^{\mathsf{T}} \lambda \geq r_1 + Cy \\ & B_1^{\mathsf{T}} \lambda \geq s_1 \end{aligned} \tag{8.16}$$

Using the dual formulation, (8.12) becomes:

$$\begin{aligned} \left(\min_{\lambda} b_1^{\mathsf{T}} \lambda + r_2^{\mathsf{T}} y \right) & \leq h \\ A_1^{\mathsf{T}} \lambda & \geq r_1 + Cy \\ B_1^{\mathsf{T}} \lambda & \geq s_1 \end{aligned}$$

Now, we use the fact that for any function ϕ and any value θ the following holds:

$$\min_x \phi(x) \leq \theta \Leftrightarrow (\exists x) \quad \phi(x) \leq \theta.$$

Finally, this leads to the following set of inequalities.

$$\begin{aligned} r_2^{\mathsf{T}} y & \leq h - b_1^{\mathsf{T}} \lambda \\ Cy & \leq A_1^{\mathsf{T}} \lambda - r_1 \\ s_1 & \leq B_1^{\mathsf{T}} \lambda \end{aligned}$$

The above inequalities define the convex set Y_h^{C}. Because its complement Y_h is not necessarily convex, we need to use its convex superset $\tilde{Y}_h$ on the given polyhedron. This is done by projecting Y_h^{C}, or its subset, onto the edges of each polyhedron as depicted in Figure 8.4

Algorithm 8.3: PolyhedronCut($\{y_1, \ldots, y_{n+1}\}, h$) returns constraint $\sigma^{\mathsf{T}} y \leq \tau$

```
// Find vertices of the polyhedron $\{y_1, \ldots, y_{n+1}\}$ inside of $Y_h^{\mathsf{C}}$
$\mathcal{I} \leftarrow \{y_i \,|\, y_i \in Y_h^{\mathsf{C}}\}$ ;
// Find vertices of the polyhedron outside of $Y_h^{\mathsf{C}}$
$\mathcal{O} \leftarrow \{y_i \,|\, y_i \in Y_h\}$ ;
// Find at least $n$ points $f_k$ in which the edge of $Y_h$ intersects an edge of
   the polyhedron
$k \leftarrow 1$ ;
for $i \in \mathcal{O}$ do
    for $j \in \mathcal{I}$ do
        $f_k \leftarrow y_j + \max_\beta\{\beta \,|\, \beta(y_i - y_j) \in (Y_h^{\mathsf{C}})\}$ ;
        $k \leftarrow k + 1$ ;
        if $k \geq n$ then
            break ;
Find $\sigma$ and $\tau$, such that $[f_1, \ldots, f_n]\sigma = \tau$ and $\mathbf{1}^{\mathsf{T}}\sigma = 1$ ;
// Determine the correct orientation of the constraint to have all $y$ in $Y_h$
   feasible
if $\exists y_j \in \mathcal{O}$, and $\sigma^{\mathsf{T}} y_j > \tau$ then
    // Reverse the constraint if it points the wrong way
    $\sigma \leftarrow -\sigma$ ;
    $\tau \leftarrow -\tau$ ;
return $\sigma^{\mathsf{T}} y \leq \tau$
```

and described in Algorithm 8.3. The algorithm returns a single constraint which cuts off part of the set Y_h^{C}. Notice that only the combination of the first n points f_k is used. In general, there may be more than n points, and any subset of points f_k of size n can be used to define a new cutting plane that constraints Y_h. This did not lead to significant improvements in our experiments. The linear program to find the pivot point with the cutting plane option is as follows:

$$\begin{aligned} \max_{\epsilon,t,y} \quad & \epsilon \\ \text{s.t.} \quad & \epsilon \leq u(Tt) - r^{\mathsf{T}} x + x^{\mathsf{T}} CTt \quad \forall x \in L \\ & \mathbf{1}^{\mathsf{T}} t = 1 \quad t \geq \mathbf{0} \\ & y = Tt \\ & \sigma^{\mathsf{T}} y \leq \tau \end{aligned} \tag{8.17}$$

Here, σ, and τ are obtained as a result of running algorithm 8.3.

Note that this approach requires that the bilinear program is in the semi-compact form to ensure that $g(y)$ is convex. The following proposition states the correctness of this procedure.

Proposition 8.16. *The resulting polyhedron produced by Algorithm 8.3 is a superset of the intersection of the polyhedron S with the complement of Y_h.*

Proof. The convexity of $g(y)$ implies that Y_h^{C} is also convex. Therefore, the intersection

$$Q = \{y \,|\, \sigma^{\mathsf{T}} y \geq \tau\} \cap S$$

is also convex. It is also a convex hull of points $f_k \in Y_h^{\mathsf{C}}$. Therefore, from the convexity of Y_h^{C}, we have that $Q \subseteq Y_h^{\mathsf{C}}$, and therefore $S - Q \supseteq Y_h$. □

8.6 Offline Bound

In this section we develop an approximation bound that depends only on the number of points for which $g(y)$ is evaluated and the structure of the problem. This kind of bound is useful in practice because it provides performance guarantees without actually solving the problem. In addition, the bound reveals which parameters of the problem influence the algorithm's performance. The bound is derived based on the maximal slope of $g(y)$ and the maximal distance among the points.

Theorem 8.17. *To achieve an approximation error of at most ϵ, the number of points to be evaluated in a regular grid with k points in every dimension must satisfy:*

$$k^n \geq \left(\frac{\|C\|_2\sqrt{n}^n}{\epsilon}\right),$$

where n is the number of dimensions of Y.

The theorem follows using basic algebraic manipulations from the following lemma.

Lemma 8.18. *Assume that for each $y_1 \in Y$ there exists $y_2 \in Y$ such that $\|y_1 - y_2\|_2 \leq \delta$ and $\tilde{g}(y_2) = g(y_2)$. Then the maximal approximation error is:*

$$\epsilon = \max_{y \in Y} g(y) - \tilde{g}(y) \leq \|C\|_2 \delta.$$

Proof. Let y_1 be a point where the maximal error is attained. This point is in Y, because this set is compact. Now, let y_2 be the closest point to y_1 in L_2 norm. Let x_1 and x_2 be the best responses for y_1 and y_2 respectively. From the definition of solution optimality we can derive:

$$\begin{aligned} r_1^{\mathsf{T}} x_1 + r_2^{\mathsf{T}} y_2 + x_1^{\mathsf{T}} C y_2 &\leq r_1^{\mathsf{T}} x_2 + r_2^{\mathsf{T}} y_2 + x_2^{\mathsf{T}} C y_2 \\ r_1^{\mathsf{T}}(x_1 - x_2) &\leq -(x_1 - x_2)^{\mathsf{T}} C y_2. \end{aligned}$$

The error now can be expressed, using the fact that $\|x_1 - x_2\|_2 \leq 1$, as:

$$\begin{aligned} \epsilon = r_1^{\mathsf{T}} x_1 + r_2^{\mathsf{T}} y_1 &+ x_1^{\mathsf{T}} C y_1 - r_1^{\mathsf{T}} x_2 - r_2^{\mathsf{T}} y_1 - x_2^{\mathsf{T}} C y_1 \\ &= r_1^{\mathsf{T}}(x_1 - x_2) + (x_1 - x_2)^{\mathsf{T}} C y_1 \\ &\leq -(x_1 - x_2)^{\mathsf{T}} C y_2 + (x_1 - x_2)^{\mathsf{T}} C y_1 \\ &\leq (x_1 - x_2)^{\mathsf{T}} C (y_1 - y_2) \\ &\leq \|y_1 - y_2\|_2 \frac{(x_1 - x_2)^{\mathsf{T}}}{\|(x_1 - x_2)\|_2} C \frac{(y_1 - y_2)}{\|y_1 - y_2\|_2} \\ &\leq \|y_1 - y_2\|_2 \max_{\{x \,|\, \|x\|_2 \leq 1\}} \max_{\{y \,|\, \|y\|_2 \leq 1\}} x^{\mathsf{T}} C y \\ &\leq \delta \|C\|_2 \end{aligned}$$

The above derivation follows from Assumption 8.3, and the bound reduces to the matrix norm using Cauchy-Schwartz inequality. □

Not surprisingly, the bound is independent of the local rewards and transition structure of the agents. Thus it in fact shows that the complexity of achieving a fixed approximation with a fixed interaction structure is linear in the problem size. However, the bounds are still exponential in the dimensionality of the space. Notice also that the bound is additive.

8.7 Contributions

There are several contributions described in this chapter. The first contribution in the chapter is the iterative algorithm that can be used to solve bilinear programs. This algorithm can be used to solve bilinear program that have a small number of bilinear terms. The second main contribution is the dimensionality reduction method that can be used to automatically reduce the number of bilinear terms based on the structure of the problem. This approach can be used to reduce the number of bilinear terms in some formulations of approximate bilinear programs.

PART III

SAMPLING, FEATURE SELECTION, AND SEARCH

CHAPTER 9

SAMPLING BOUNDS

Sampling is an important part of value function approximation. The remainder of the thesis generally assumes that the sampled states and actions satisfy Assumptions 2.26, 2.27, and 2.28, which are quite general. This chapter describes structures of MDPs than can be used to satisfy these assumptions. We establish both worst-case and probabilistic bounds that leverage existing machine learning regression methods. These bounds rely on the guarantees of optimization–based algorithms for value function approximation coupled and the regularization of the value function.

The sampling bounds in this chapter apply to domains in which samples cannot be gathered at all. Instead, they are provided in advance and the solution must be controlled for the missing ones. For example, the bounds can be used to choose the appropriate features, as discussed in Chapter 10.

The sampling bounds we proposed could also apply to domains with an available model, in which samples can be gathered in an arbitrary order for any state and action. Because it may be expensive — computationally or otherwise — to gather samples, it is still necessary to use as few of them as possible. The methods that we propose can be used to greedily choose samples that minimize the bounds. It may be, however, more interesting to develop methods that minimize the sampling error globally.

In comparison to previous work, the bounds proposed in this chapter explicitly consider value function approximation and can be used to derive bounds on the policy loss. In particular, to obtain bounds on the policy loss are obtained in conjunction with Theorem 4.4 and Theorem 5.13, depending on the value function approximation method used.

This chapter is organized as follows. First, Section 9.1 outlines the basic considerations and the framework we consider in developing the sampling bounds. As discussed in Section 2.5, the sampling error consists of two main components: 1) the state selection error, and 2) the transition estimation error. Section 9.2 presents bounds on the state selection error (Assumption 2.26). Section 9.3 analyzes the errors due to the estimation of the initial distribution (Assumption 2.27). Section 9.4 presents bounds on transition estimation error (Assumption 2.28) and proposes to use common random numbers to reduce the error. Then, Section 9.5 shows how the sampling framework can be implemented. Section 9.6 draws connections to other work on sampling considerations in value function approximation and compares it to the proposed approach. Finally, Section 9.7 presents experimental evaluation of the sampling methods.

9.1 Sampling In Value Function Approximation

Sampling, as an optimization with incomplete information, is an issue that has been widely addressed in the machine learning community for the problems of regression and classification. The challenge in these problems is to guarantee that a solution based on a small subset of the data generalizes also to the remainder of the data (Devroye, Gyorfi, & Lugosi, 1996; Vapnik, 1999; Bousquet, Boucheron, & Lugosi, 2004).

The goal of regression in machine learning is to estimate a function on a domain from a set of its values. We now formulate value function approximation as a regression problem in order to illustrate the differences. Our treatment of regression here is very informal; please see for example Gyorfi, Kohler, Lrzyzak, and Walk (2002) for a more formal description. Regression in value function approximation can be seen as the problem of estimating a function $f : \mathcal{S} \to \mathbb{R}$ that represents the Bellman residual:

$$f(s) = (v - Lv)(s)$$

for some *fixed* value function v. Regression methods typically assume that a subset of the function values is known. That is why $f(s)$ must represent the Bellman residual — which is known from the samples — and not the value function — which is unknown.

Again, the goal of regression is to find a function $\tilde{f} \in \mathcal{F}$ based on sampled values of $f(s_i)$ for $s_1 \ldots s_n \in \mathcal{S}$ drawn i.i.d. according to a distribution μ. Commonly, the function $\tilde{f}$ is chosen by minimizing the sample error $\sum_{i=1}^{n}(f(s_i) - \tilde{f}(s_i))^2$ to minimize the true error $\sum_{s\in\mathcal{S}}(f(s) - \tilde{f}(s))^2$. The sampling bounds are then on the difference between the sample error and the true error:

$$\left| \frac{1}{n} \sum_{i=1}^{n} (f(s_i) - \tilde{f}(s_i))^2 - \sum_{s\in\mathcal{S}} \mu(s)(f(s) - \tilde{f}(s))^2 \right|.$$

Most bounds on this error rely on the redundancy of the samples with respect to the set of functions in $\mathcal{F}$. That is, given a sufficient number of samples $s_1 \ldots s_n$, additional samples have very little influence on which function $\tilde{f}$ is chosen from the set of possibilities.

The problem with value function approximation is that there is no fixed distribution μ over the states that limits the importance of function values. An important distribution over the states in bounding the policy loss is the state visitation frequency $(1 - \gamma)u_\pi$ for a policy π. Here, $(1 - \gamma)$ is just a normalization coefficient. Unfortunately, this distribution depends on the policy, which depends on the value function. In the regression, this would mean that μ is not fixed, but it instead depends on the result $\tilde{f}$. As a result, at no point it is simply possible to assume that additional samples will have a small influence on the choice of the function $\tilde{f}$ — the cumulative distribution may shift heavily in favor of the non-sampled states with every new sample.

Because of the difference between regression and value function approximation setting, we need different assumptions than the assumptions that are standard in regression. In particular, the assumptions in regression do not concern the space which is not sampled, since most likely, it is not likely according to μ. On the other hand, the assumptions for sampling bounds for value function approximation must be uniform over the state space, since the distribution μ may change arbitrarily. Therefore, the assumptions for value function approximation must be stronger. Yet, as we show later in the chapter, regression results methods *can* be used to compute tight bounds on the sampling error.

To derive bounds on the sampling error we assume that the representable value functions are regularized. Our focus is on regularization that uses the L_1 norm, but the extension to other polynomial norms is trivial using the Holder's inequality. The advantages of the L_1 norm are that 1) it can be easily represented using a set of linear constraints, and 2) it encourages sparsity of the solution coefficients. We use the following refinement of Assumption 2.21.

Assumption 9.1. The set of representable functions for the L_1 norm are defined as:

$$\mathcal{M}(\psi) = \{\Phi x \mid \|x\|_{1,e} \leq \psi\}.$$

such that $\phi_1 = \mathbf{1}$ and $e(1) = 0$ and $e(i) > 0$ for all $i > 1$. Note that the norm is weighted by e.

Assumption 9.1 implies Assumption 2.21. We also use the following weighted L_∞ norm:

$$\|x\|_{\infty,e_{-1}} = \max_i \begin{cases} |x(i)|/e(i) & \text{when } |x(i)| > 0 \\ 0 & \text{otherwise} \end{cases}.$$

The following lemma relates the weighted L_1 and L_∞ norm and will be useful in deriving bounds.

Lemma 9.2. *Let $v \in \mathcal{M}(\psi)$ as defined in Assumption 9.1. Then for any y of an appropriate size:*

$$|y^\mathsf{T} v| \leq |y|^\mathsf{T} |v| \leq \psi \|y\|_{\infty,e_{-1}}$$

The assumptions that we introduce must capture the structure of the MDP. Because many structures have been studied in the context of metric spaces, it is convenient to map the state space to a metric space (in particular $\mathbb{R}^n$). This makes it possible to take advantage of the structures proposed for metric spaces.

Definition 9.3 (State Embedding Function)**.** The state embedding function $k : \mathcal{S} \to \mathbb{R}^n$ maps states to a vector space with an arbitrary norm $\|\cdot\|$. The state–action embedding function $k : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}^n$ maps states and actions the a vector space with an arbitrary norm $\|\cdot\|$. It is also possible to define a state–action–feature embedding function similarly.

The state embedding function is a part of the assumed structure of the MDP. In many applications, the definition of the function follows naturally from the problem. The meaning of the function k is specified when not apparent from the context.

9.2 State Selection Error Bounds

The state selection error — defined in Assumption 2.26 — is due to value function error in states that are not covered by the samples. With no assumptions over the structure of the transitions and rewards over the state space, the error may be arbitrarily large. Even simply bounding the maximal and minimal reward does not lead to useful bounds. It is, therefore, necessary to propose structures that can be used to bound the sampling error.

In this thesis, we are interested in problems with very large, or infinite state spaces. To avoid unnecessary technical issues, we assume that the state space is finite, but very large. Even when the state space is very large, it may be possible that the optimal policy does not reach a large number of states from the initial states. It is then unnecessary to bound the approximation error on states that are not visited. Methods that are suitable for such domains are discussed in more detail in Section 9.6 and Chapter 11. In this chapter, we assume that any policy will visit a very large number of states — much larger than it is possible to sample.

The state selection error bound assumes that the transitions for the sampled states are known precisely — i.e. samples $\bar{\Sigma}$ are available. The transition estimation error for samples $\tilde{\Sigma}$ is additive and can be treated independently.

One of the main difficulties in deriving the sampling bounds is to capture use the fact that the value functions are transitive–feasible with respect to the *sampled* Bellman residual. More formally, this means that it is necessary to guarantee the following:

$$\min_{s \in \mathcal{S}} (v - Lv)(s) \geq -\epsilon_p,$$

or also the following:

$$\min_{s \in \mathcal{S}} (v - Lv)(s) - \min_{s \in \bar{\Sigma}} (v - \bar{L}v)(s) \geq -\epsilon_p.$$

We propose two methods that can capture this fact. We start with a simple *local modeling assumption* and then extend it to regression-based models.

9.2.1 Local Modeling Assumption

The local modeling assumption assumes that states that are close in the embedded space are also similar in terms of transitions and rewards. The idea is to define a mapping that maps every state to a *single* similar sampled state. Because of the similarity to the sampled states, the Bellman constraint violation in the unsampled state cannot be very large, as we show formally below. The following theorem shows how mapping *every* state to a sampled state can bound the state selection error.

Theorem 9.4. *Assume Assumption 9.1 and that there is a sample mapping function $\chi : \mathcal{S} \to \bar{\Sigma}$ such that for all $v \in \mathcal{M}(\psi)$:*

$$\max_{s \in \mathcal{S}} |(v - Lv)(s) - (v - \bar{L}v)(\chi(s))| \leq \epsilon(\psi).$$

Then, the state selection error in Assumption 2.26 can be bounded as: $\epsilon_p(\psi) \leq \epsilon(\psi)$.

The proof of the theorem can be found in Section C.10. The actual local modeling assumption is as follows.

Assumption 9.5. Assume that samples $\bar{\Sigma}$ are available. Assume also that features and transitions satisfy Lipschitz-type constraints:

$$\begin{aligned} \|\phi(\bar{s}) - \phi(s)\|_{\infty, e_{-1}} &\leq K_\phi \|k(s) - k(\bar{s})\| \\ |r(\bar{s}) - r(s)| &\leq K_r \|k(s) - k(\bar{s})\| \\ \|P(\bar{s}, a)^\mathsf{T} \phi_i - P(s, a)^\mathsf{T} \phi_i\|_{\infty, e_{-1}} &\leq K_p \|k(s) - k(\bar{s})\| \quad \forall a \in \mathcal{A} \end{aligned}$$

Here ϕ_i denotes a vector representation of a feature across all states.

The assumption states that the parameters of the MDP are Lipschitz continuous and is very similar to the local modeling assumption in metric E^3 (Kakade, 2003; Kakade, Kearns, & Langford, 2003). An alternative assumption with almost identical implications would bound the difference $\|P(\bar{s}, a) - P(s, a)\|_1$.

Assumption 9.5 characterizes the general properties of the MDP, but ignores the properties of the actual sample. The following assumption unifies the assumptions on the MDP and the sampling in terms of mapping states to sampled states.

Assumption 9.6 (Sufficient Sampling)**.** Assume a function $\chi : \mathcal{S} \to \bar{\Sigma}$ such that for all $s \in \mathcal{S}$:

$$\begin{aligned} \|\phi(\chi(s)) - \phi(s)\|_{\infty, e_{-1}} &\leq \sigma_\phi \\ |r(\chi(s)) - r(s)| &\leq \sigma_r \\ \|P(\chi(s), a)^{\mathsf{T}} \phi_i - P(s, a)^{\mathsf{T}} \phi_i\|_{\infty, e_{-1}} &\leq \sigma_p \quad \forall a \in \mathcal{A} \end{aligned}$$

We define this intermediate assumption because it is helpful in establishing bounds on ϵ_c, which are described later in this chapter. Assumption 9.6 is used to prove the following proposition.

Proposition 9.7. *Assume Assumption 9.1, Assumption 9.5, and a state mapping function*

$$\chi(s) = \arg \min_{\bar{s} \in \bar{\Sigma}} \|k(s) - k(\bar{s})\|.$$

such that $\max_{s \in \mathcal{S}} \|k(s) - k(\chi(s))\| \leq q$ *for some* $q \in \mathbb{R}$*. Then, Assumption 2.26 holds with the following constant:*

$$\epsilon_p(\psi) = qK_r + q\psi(K_\phi + \gamma K_p).$$

The proof of the proposition can be found in Section C.10. The importance of this bound is that the state sampling error grows linearly with the increasing coefficient ψ. This bound does not address methods for computing $\arg\min_{\bar{s} \in \bar{\Sigma}} \|k(s) - k(\bar{s})\|$. The simplest method of actually computing this value would be to use a triangulation of the available samples $k(\Sigma)$. Because of the limited practical utility of this bound, we do not discuss these technical details.

9.2.2 Regression Bounds

The difficulty with the local modeling assumption is exactly that it is local; it assumes the worst case and does not consider the global properties of the MDP parameters. Here, we propose a more sophisticated method for estimating the sampling error, which does not rely on the locality assumption, but instead utilizes machine learning regression methods.

To develop better methods, we first need to extend Theorem 9.4. This extension will show that as long as the parameters of the MDP are in some sense linear in states between the samples, there is no state selection error. Figure 9.1 compares this assumption with the local modelling assumption. The circles represent sampled values, the shaded cones represent the uncertainty using the local model, and the line represents values that do not

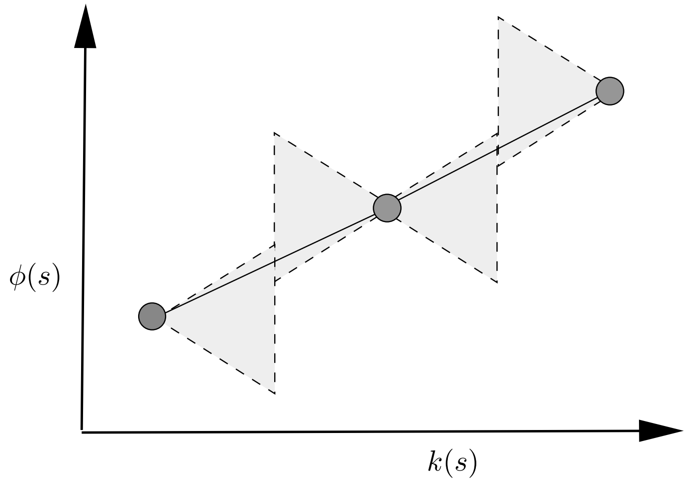

Figure 9.1. Value of a feature as a function of the embedded value of a state.

incur any error. Because it is unlikely that the parameters are truly linear, we show that the error can be quantified by the deviation from the linearity.

Here, we focus on providing general guidelines; the actual implementation relies on Gaussian processes and is discussed in Section 9.5. The following theorem extends Theorem 9.4 to utilize existing regression methods.

Theorem 9.8. *Assume Assumption 9.1 and a sample mapping function $\chi : (\mathcal{S} \times \mathcal{A}) \to \bar{\Sigma}^{n+1}$, where n is the dimensionality of the state–action embedding function $k(\mathcal{S}, \mathcal{A})$ and $\bar{\Sigma}^{n+1}$ denotes subsets of length $n + 1$. The sample mapping function χ satisfies for all $s, a \in \mathcal{S}, \mathcal{A}$:*

$$\begin{aligned} k(s,a) &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) k(s_i,a_i) \\ 1 &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) \end{aligned}$$

for some function $\beta_i(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ and in addition for some $\sigma_\phi, \sigma_r \in \mathbb{R}$:

$$\begin{aligned} \left\| \left(\phi(s) - \gamma P(s,a)^\mathsf{T} \Phi\right) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) \left(\phi(s_i) - \gamma P(s_i,a_i)^\mathsf{T} \Phi\right) \right\|_{\infty, e_{-1}} &\leq \sigma_\phi \\ \left| r(s,a) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) r(s_i,a_i) \right| &\leq \sigma_r. \end{aligned}$$

Then, the following holds:

$$\epsilon_p(\psi) \leq \sigma_r + \sigma_\phi \psi.$$

The proof of the theorem can be found in Section C.10. There are a few important differences between Theorem 9.4 and Theorem 9.8. The crucial one is that χ in Theorem 9.8

maps a state to multiple samples, unlike the one Theorem 9.4 which maps every state to a single sample. The difference between defining the function χ for states or states and actions is only a superficial one.

Theorem 9.8 applies when the assumption is satisfied with certainty. Worst–case guarantees are often too strict. It may be desirable to allow a limited chance of failure in order to improve the mean performance. In addition, most regression methods only probabilistic guarantees. The following corollary trivially extends the theorem above to the probabilistic case.

Corollary 9.9. *Assume Assumption 9.1 and that the hypothesis of Theorem 9.8 is satisfied with probability* $1 - \delta$ *for some* $\delta \in (0, 1)$ *. Then:*

$$\mathbf{P}\left[\epsilon_p(\psi) \le \sigma_r + \sigma_\phi \psi\right] \ge 1 - \delta.$$

The probability here is with respect to the sample and *the prior expectation over the model.*

The MDP can then be estimated using standard regression methods for the following two functions:

$$\begin{aligned} \rho_1(s, a, i) &= \phi_i(s) - \gamma P(s, a)^\mathsf{T} \phi_i \\ \rho_2(s, a) &= r(s, a) \end{aligned}$$

This is a form of specialized Bellman residual for the features and the rewards. It is important that these values do not depend on the value function and can be estimated directly from the data. To make the regression model useful, it needs to satisfy the following assumption.

Assumption 9.10. Let $\tilde{\rho}_1$ and $\tilde{\rho}_2$ be the output of the regression algorithm on $\bar{\Sigma}$ for the following functions:

$$\begin{aligned} \rho_1(s, a, i) &= \phi_i(s) - \gamma P(s, a)^\mathsf{T} \phi_i \\ \rho_2(s, a) &= r(s, a). \end{aligned}$$

Then, these functions must satisfy that:

$$\mathbf{P}\left[\max_{s \in \mathcal{S}, a \in \mathcal{A}, i \in 1 \ldots |\phi|} |\tilde{\rho}_1(s, a, i) - \rho_1(s, a, i)| + |\tilde{\rho}_2(s, a) - \rho_2(s, a)| - \epsilon(s, a, i) < 0\right] \ge 1 - \delta.$$

The random variables in this probability are $\tilde{\rho}_1$ and $\tilde{\rho}_2$.

This requirement is similar to the standard PAC learning bounds, but differs in some crucial ways. In particular, the PAC bounds usually require that :

$$\mathbf{P}\left[\mathbf{E}\left[|\tilde{\rho}_1(s, a, i) - \rho_1(s, a, i)|\right] < \epsilon\right] \ge 1 - \delta.$$

This bounds the expected error instead of the maximal error. This assumption translates to the state selection error as the following corollary shows.

Corollary 9.11. *Assume a sample mapping function $\chi : (\mathcal{S} \times \mathcal{A}) \to \bar{\Sigma}^{n+1}$, where n is the dimensionality of the state–action embedding function $k(\mathcal{S}, \mathcal{A})$ and $\bar{\Sigma}^{n+1}$ denotes subsets of length $n+1$. The sample mapping function χ satisfies for all $s, a \in \mathcal{S}, \mathcal{A}$:*

$$\begin{aligned} k(s,a) &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) k(s_i,a_i) \\ 1 &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) \end{aligned}$$

for some function $\beta_i(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ and in addition for some $\sigma_\phi, \sigma_r \in \mathbb{R}$ with probability $1-\delta$:

$$\max_{j=1\ldots|\phi|} \left| \tilde{\rho}_1(s,a,j) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a)\tilde{\rho}_1(s_a,a_i,j) \right| \le \sigma_\phi$$

$$\left| \tilde{\rho}_2(s,a) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a)\tilde{\rho}_2(s_i,a_i) \right| \le \sigma_r.$$

Then, with probability $1-\delta$, the following holds:

$$\epsilon_p(\psi) \le \sigma_r + \sigma_\phi \psi.$$

To actually implement the bounds from this section, it is necessary to define the regression method for learning ρ and also to define the sample mapping function χ. There are many options for implementing them; we show one possibility in Section 9.5.

9.3 Uniform Sampling Behavior

In this section, we discuss methods that bounds the error due to the imprecise estimation of the ALP initial distribution — formally defined in Assumption 2.27. This error is not very important, because most methods that we study do not use the initial distribution and if they do, it is typically a single state — that is easy to estimated. The initial distribution is, for example, needed in some ALP formulations. Because this choice is inherently heuristic, it is often unimportant how precise the estimation is. Our analysis is, therefore, quite limited and shows mostly that a small error in the estimation results in a small error in the value function.

These bounds require additional assumptions on the sampling procedure. That is the sampling must not only cover the whole space, but also must be uniformly distributed over it. In addition, we have to rely on a form of the local modelling assumption to ensure that a crucial state of the distribution is not missing. The actual uniformity assumption follows.

Assumption 9.12 (Uniform Sampling)**.** Assume Assumption 9.6 and the corresponding function χ. Let $\chi^{-1} : \bar{\Sigma} \to 2^{\mathcal{S}}$ be the inverse function that represents the closest states in $\mathcal{S}$ to samples $\bar{\Sigma}$. Assume that for all $\bar{s} \in \bar{\Sigma}$:

$$\frac{|\mathcal{S}|}{|\bar{\Sigma}|}(1-\sigma_s) \le |\chi^{-1}(\bar{s})| \le \frac{|\mathcal{S}|}{|\bar{\Sigma}|}(1+\sigma_s)$$

This assumption requires that the sets of closest states for all samples are of similar size. This assumption can be satisfied only in specific situations, which we do not analyze here. The following theorem shows the implication that the assumption can guarantee the desirable distribution estimation behavior.

Theorem 9.13. *Assume Assumption 9.1, Assumption 9.6, and Assumption 9.12 and let $v \in \mathcal{M}(\psi)$ be a representable value function. Then, Assumption 2.27 is satisfied with the following constant:*

$$\epsilon_c = \sigma_s \|\Phi^\mathsf{T} \bar{c}\|_{\infty, e_{-1}} \psi + 2\sigma_\phi \psi$$

when $c = \alpha$ and $e = (0 \quad \mathbf{1}^\mathsf{T})^\mathsf{T}$.

The proof of the theorem can be found in Section C.10.

This error bound is somewhat related to the transition estimation error. The transition estimation error bounds the error in estimating the transitions from the states and may be very important in many domains; the bounds on the transition estimation error follow.

9.4 Transition Estimation Error

This section shows how to bound the transition estimation error — formally defined in Assumption 2.28. This error is due to states that are included in the samples, but their transition probabilities are estimated imprecisely.

We first propose simple bounds on the transition estimation error. These simple bounds, without assuming any special structure, turn out to depend linearly on the number of samples, which can be very large. As we also show, these bounds are asymptotically tight and therefore indicate that the transition estimation error may be very high in some cases.

To reduce the transition estimation error and the corresponding bound, we propose to use common random numbers — a technique popular in some simulation communities — to tighten the bounds and improve the performance (Glasserman & Yao, 1992).

The transition estimation error bounds hold only with a limited probability; is it always passible that the sample is misleading and that the transition probabilities are computed incorrectly. We demonstrate the importance of the transition estimation error in the reservoir management problem in Section 9.7.

The transition estimation error ϵ_s, as we treat it, is independent of the state sampling error ϵ_p — that is the errors are additive. This section, therefore, assumes that for any $(s, a) \in \tilde{\Sigma}$ there is a $(s, a) \in \bar{\Sigma}$, and vice versa.

Theorem 9.14. *Assume that samples $\tilde{\Sigma}$ are available and that the number of samples per each state and action pair is at least n. Also assume that $e = (0\mathbf{1}^\mathsf{T})^\mathsf{T}$. Then, the transition estimation error ϵ_s (Assumption 2.28) is bounded as:*

$$\begin{aligned}\mathbf{P}\left[\epsilon_s(\psi) > \epsilon\right] &\le Q\left(2|\phi| \exp\left(\frac{2(\epsilon/(\psi \cdot \gamma))^2}{M_\phi n}\right), |\tilde{\Sigma}|_a\right) \\ &\le 2|\tilde{\Sigma}|_a |\phi| \exp\left(\frac{2(\epsilon/(\psi \cdot \gamma))^2}{M_\phi n}\right),\end{aligned}$$

where $|\tilde{\Sigma}|_a$ *is the number of sampled state-action pairs,*

$$M_\phi = \max_{s \in \mathcal{S}} \|\phi(s)\|_\infty.$$

and

$$Q(x, y) = 1 - (1 - x)^y.$$

The proof of the theorem can be found in Section C.10. The proof of the theorem closely resembles the proofs in statistical learning theory (Bousquet et al., 2004) with one crucial difference. In statistical learning theory, the approximation error must be bound for the set of all potential classification or regression functions, just as we need to bound the error over all sampled states and actions. The important difference is in the dependence of the errors. Because the classifiers are evaluated using the same samples, the error over them are potentially dependent. In our setting, the samples are gathered independently for each sampled state and this independence leads to somewhat tighter bounds using the function Q instead of the union bound.

The function Q grows close to linearly in y when x is close to 0. A significant weakness of this bound is that it depends, close to linearly, on the number of samples and features. While the number of features is often small, the number of samples is usually very large. As a result, the bound can be very loose. Unfortunately, the transition estimation error is actually inevitable and is not only due to the looseness of the bounds as the following shows.

Proposition 9.15. *The bound in Theorem 9.14 is asymptotically tight with respect to* $|\tilde{\Sigma}|_a$.

The proof of the proposition can be found in Section C.10. Interestingly, the bounds in Theorem 9.14 show that there is a tradeoff between the state selection error and the transition estimation error. With an increasing number of states sampled, the state selection error tends to decrease. However, the bounds on the transition estimation error will increase, when the number of transitions estimated per state remains the same. We show next that sometimes this tradeoff can be circumvented by deriving bounds that are independent of the number of samples.

9.4.1 Common Random Numbers

The results above indicate that the transition estimation error may be very large when the number of sampled states and actions is large. The transition estimation error, and the corresponding bound, can be decreased by assuming that the Bellman residual of the solution is close to 0 for a small number of states and actions. This is, however, not true in general. Here, we instead propose to use *common random numbers* to estimate transitions; this approach is applicable to a large class of industrial problems.

Common random numbers are applicable to problems with external uncertainty, which is not affected by the actions taken. We use the random variable $\omega \in \Omega$ to denote this external uncertainty for some. In the reservoir management problem (see Section B.3), the variable ω may represent the weather — an external variable that cannot be influenced. In blood inventory management (see Section B.2), the variable ω represents the external uncertainty of supplies and demands, which is independent of the current state of the inventory.

We now specify how the state transition probabilities depend on the external source stochasticity. Let the random variable ω be defined such that:

$$P(s, a, z(s, a, \omega)) = \mathbf{P}[\omega] \qquad \forall s \in \mathcal{S}, a \in \mathcal{A},$$

where $z : \mathcal{S} \times \mathcal{A} \times \Omega \to \mathcal{S}$ is the deterministic transition function. This definition applies to discrete ω; the definition for the continuous case would be similar.

The ordinary samples in this setting would are defined as:

$$\tilde{\Sigma} \subseteq (s, a, (s_1 \dots s_n), r(s, a)) \,|\, s, s' \in \mathcal{S},\ a \in \mathcal{A}\},$$

where $s_1 \dots s_n$ are selected i.i.d. from the distribution $P(s, a)$ for every s, a independently. The *common random number* samples are denoted as $\tilde{\Sigma}_c$ and defined as:

$$\tilde{\Sigma}_c \subseteq (s, a, (z(s, a, \omega_1) \dots z(s, a, \omega_n)), r(s, a)) \,|\, s, s' \in \mathcal{S},\ a \in \mathcal{A}\},$$

where $\omega_1 \dots \omega_n \in \Omega$ are selected i.i.d. according to the appropriate distribution. This distribution is *independent* of the current state and action. The common random number samples are used in the construction of ABP and ALP identically as the regular samples.

Practically, using the common random numbers in the reservoir management means that weather is sampled independently of the level of the water in the reservoir. The constraints are then constructed for each water level, averaging the effects of the weather. Intuitively, this is desirable because the comparison between the values of various water levels are more fair when the weather is assumed to be identical.

To simplify the notation, we use

$$X_i(\omega_j) = |P(s_i, a_i)^{\mathsf{T}} \phi_f - P(s_i, a_i, z(s_i, a_i, \omega_j))^{\mathsf{T}} \phi_f|$$

for *some* feature f and $i \in \tilde{\Sigma}$. These random variables are not necessarily independent, as they would be if the state transitions were sampled *independently*. This random variables are have unknown values for the samples, since $P(s_i, a_i)^{\mathsf{T}} \phi_f$ is not known. The random variables X_i are, however, useful for theoretical analysis.

The following definition of a growth set defines a suitable structure for bounding the transition estimation error for common random numbers.

Definition 9.16 (Growth Set)**.** The growth set for the number of states m is defined as:

$$\tau(m) = \{s \in \mathcal{S} \,|\, \omega_j \in \Omega,\ X_i(\omega_j) \geq X_{i'}(\omega_j),\ i \in \mathcal{S}\}.$$

Note that the definition is for all states, not only the sampled ones, although it could be defined for the samples only.

The purpose the definition of the growth set is more explanatory than practically useful at this point. It is currently a longstanding challenge in statistical learning theory to derive measures of complexity that both lead to sharp upper bounds and can be easily computed for a class of problems (Bousquet et al., 2004; Devroye et al., 1996). While the proposed growth set can be roughly estimated in some domains, it cannot be derived from the samples alone.

The following theorem shows the theoretical importance of using the common random numbers. The important difference is that the bound does not depend on the number of sampled states, but on the growth function instead.

Theorem 9.17. *Assume that samples $\tilde{\Sigma}$ are generated using* common random numbers *and that the number samples of common numbers is m. Then:*

$$\{\mathrm{I}\{z(s,a,\omega)v \geq \epsilon\} \mid \omega \in \Omega, v \in \mathcal{M}(\psi), \epsilon \in \mathbb{R}_+\}.$$

Then, the transition estimation error ϵ_s (Assumption 2.28) is bounded as:

$$\mathbf{P}\left[\epsilon_s(\psi) > \epsilon\right] \leq 2|\phi| \exp\left(\frac{2(\epsilon/(\psi \cdot \gamma \cdot |\tau(m)|))^2}{M_\phi m}\right),$$

where $|\tilde{\Sigma}|_a$ is the number of sampled state-action pairs and

$$M_\phi = \max_{s \in \mathcal{S}} \|\phi(s)\|_\infty.$$

The proof of the theorem can be found in Section C.10. The common random numbers in samples introduce correlation among the constraints, which introduces additional structure into the problem. This can be then used to bound the error.

It is not always trivial to determine the growth function of the set of functions. It is important to study the growth functions in the particular domains, but it may be also possible to derive complexity measures that can be estimated directly from the sample. Such an example for classification problems is the Rademacher complexity (Bousquet et al., 2004).

9.5 Implementation of the State Selection Bounds

This section describes a specific approach for bounding the state sampling error. Section 9.2 shows just a general framework which can be used with an arbitrary regression method. In particular, we propose to use Gaussian processes (Rasmussen & Williams, 2006) which can be used to satisfy all the necessary requirements.

Gaussian processes represent a general *Bayesian* method for solving general regression and classification problems. These methods assume an a priori distribution over the value functions and compute a posteriori estimation based on the samples. The initial and a posteriori distribution are multivariate Normal distributions — hence the name. Gaussian process, in particular, is an infinitely-dimensional Normal distribution. Technically, the MDPs we consider are all finite, but since the structure of the problem is computed in the embedded real space, it is necessary to use infinitely–dimensional distributions.

The use of a Gaussian process implies that the assumptions on the parameters of the MDP are defined in terms of prior beliefs. These beliefs do not include simply the mean and the variance of the features and rewards, but also crucially the covariance. The covariance function among states defines the generalization properties. There are many possibilities for choosing these covariance functions, or kernels (Rasmussen & Williams, 2006), and we discuss them in greater detail below.

It is important to note that there is no inherent reason why the values used to estimate the Bellman residual should be normally distributed. The Normal distribution in our setting simply represents the *prior* belief on the parameter values. However, when only $\tilde{\Sigma}$ are available, it may be necessary to model the *estimation* error using a normal distribution. There is some justification for this, as we discuss below.

The following assumption captures the assumptions necessary to apply the Gaussian process for estimating the parameters of the MDP.

Assumption 9.18. Let $\kappa_\phi : (\mathcal{S}, \mathcal{A}, \mathbb{N}_{|\phi|}) \times (\mathcal{S}, \mathcal{A}, \mathbb{N}_{|\phi|}) \to \mathbb{R}$ represent the covariance function for the feature values and let $\kappa_r : (\mathcal{S}, \mathcal{A}) \times (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ represent the covariance function for the rewards. The values parameters of the MDP are distributed according to the following distributions:

$$\begin{aligned} P(s,a)\phi_i &= \rho_1 \sim \mathcal{N}(\mathbf{0}, \kappa_\phi) \\ r(s,a) &= \rho_2 \sim \mathcal{N}(\mathbf{0}, \kappa_r) \end{aligned}$$

where κ_ϕ and κ_r represent positive semidefinite covariance functions.

The assumption implies that the feature values are independent of the rewards for the states and actions. This is not a requirement, and if it makes sense for the domain, the dependence may be included. We also assume that the variance of the function value is zero, because the samples available are $\bar{\Sigma}$. We discuss how to combine these results with bounds on the estimation error later in this section.

The important property of Gaussian processes is that it is possible, and easy, to bound the probability of a significant deviation from the estimated mean. That is to derive the bounds, it is necessary to provide bounds on:

$$\mathbf{P}\left[\max_{s\in\mathcal{S},a\in\mathcal{A},i} |\tilde{\rho}_1(s,a,i) - \rho_1(s,a,i)| + |\tilde{\rho}_2(s,a) - \rho_2(s,a)| < \epsilon\right] \geq 1-\delta.$$

These are standard bounds, which can be found in the literature (Rasmussen & Williams, 2006). In the remainder of the section, we assume that the a posteriori distribution, given the samples is available.

To use Theorem 9.8, we must define the sample mapping function χ. One simple option is to triangulate the samples and we assume that the state space is triangulated. While using a triangulation may potentially loosen the bounds, it is in general convenient to deal with. Let $T = \{T_1 \dots T_m\}$ denote a triangulation of the set $k(\bar{\Sigma})$ where T_i represent the polyhedrons. To simplify notation, we use $T_i(j)$ to refer to the *state–action sample* s, a of the j-th vertex of the i-th polyhedron, not its embedded value $k(s,a)$. The function $\chi(s,a)$ can now be defined as follows.

$$\chi(s,a) = \{T_i(1) \dots T_i(n+1)\} \qquad \text{where } k(s,a) \in T_i.$$

That is, every state and action are mapped to the vertices of the polyhedron the sample is contained in.

The following proposition is a trivial consequence of the analysis above and shows how Gaussian processes can be used in our context.

Proposition 9.19. *Assume a triangulation $T = \{T_1 \ldots T_m\}$ of $k(\mathcal{S})$ where n is the dimension of $k(\mathcal{S})$. In addition, assume that $\tilde{\rho}_1$ and $\tilde{\rho}_2$ are Gaussian processes with* a posteriori *$1-\delta$ confidence lower bounds l_1, l_2 and upper bounds u_1, u_2. Note that the confidence bounds on a set of values is not the same as the union of individual confidence bounds. Then, the hypothesis of Theorem 9.8 holds with the minimal possible value of the following constants:*

$$\sigma_\phi \geq \max_{T_j \in T} \max_{i \in |\phi|} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_1(T_j(l), i) - l_1\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right)$$

$$\sigma_\phi \geq \max_{T_j \in T} \max_{i \in |\phi|} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} u_1\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right) - \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_1(T_j(l), i)$$

$$\sigma_r \geq \max_{T_j \in T} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_2(T_j(l)) - l_2\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right)$$

$$\sigma_r \geq \max_{T_j \in T} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} u_2\left(\sum_{l=1}^{n+1} \beta_l T_j(l)\right) - \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_2(T_j(l))$$

where $B = \{\beta_1 \ldots \beta_{n+1} \,|\, \sum_{l=1}^{n+1} \beta_l = 1, \beta_l \geq 0\}$.

The proof of the proposition can be found in Section C.10. Informally, the bound is in terms of the maximal deviation of the approximated function from the piecewise linear approximation from the samples.

The method for finding the minimum and maximum of the mean of the Gaussian process and also the maximum deviation with confidence δ is independent of the approach that we propose. It depends in particular in the actual covariance function that is used. We provide details on some covariance functions in Section 9.7

So far, we have assumed that the more informative samples, containing the distribution, were available in computing the transitions. This is not always the case and it may be necessary for ρ_1, ρ_2 to also consider the transition estimation error. This can be achieved by changing the a priori distribution to have non-zero variance for value functions. A careful analysis of such a case is not yet available.

9.6 Discussion and Related Work

Sampling in MDPs has been studied extensively. The greatest focus has been, however, on methods for finite-state MDPs. These approaches cannot gather samples arbitrarily, but instead must act in the environment. The penalty for gathering samples is the lost opportunity of acting optimally. Because these approaches do not use value function approximation — the state space is assumed to be sufficiently small for precisely computing the value — they are not directly related to the sampling methods proposed in this thesis.

The tradeoff between exploration — that is sampling to learn the environment — and exploration — that is collecting reward — is a very important study in reinforcement learning. Below, we provide a *non-exhaustive* list of approaches that trade off exploration with exploitation. Other significant methods have been proposed.

E^3 An algorithm that explicitly explores the state space for unknown states. These results require some limited "mixing" rate of the MDP (Kearns & Singh, 1998) and apply to both discounted and average reward objectives.

R-max A simplified version of E^3 algorithm (Brafman & Tennenholtz, 2002). It also extends the analysis to stochastic games. The exploration in the algorithm is implicit through assuming that the reward in unexplored states is the largest possible (hence the name: R-max).

Metric E^3 Extends E^3 to metric spaces to provide generalization (Kakade et al., 2003). This algorithm uses the *local modeling assumption*, which is closely related to the assumptions that we make.

UCRL2 Unlike the other two algorithms, they also consider the distribution probabilities when an action is taken in an "unknown" state. That is instead of simply two possible modes (exploration and exploitation), there is a continuum. This leads to better bounds on the regret. It also does not require mixing rates on MDPs but uses a notion of a diameter (Auer, Jaksch, & Ortner, 2009). This is an extension of the previous version: UCRL.

MBIE Similar to R-max, but it interleaves value iteration with the exploration (Strehl & Littman, 2005, 2008). Most other algorithms assume that the policy is completely recomputed after exploring a state, while MBIE only updates values of a small number of states.

BOSS Similar to MBIE and UCRL2, but also adds a Bayes bias to speed up learning. Samples many MDPs and uses an extended value iteration to compute the maximum over multiple possible sampled MDPs (Asmuth, Li, Littman, Nouri, & Wingate, 2009).

CORL Learns a parametric model of the MDP from acting in the environment (Brunskill, Leffler, Li, Littman, & Roy, 2009). This model addresses a limited — albeit quite interesting — class of MDPs with an infinite state space. Unlike, other methods, this algorithm also considers value function approximation. The approximation is treated only implicitly through assumptions on the approximate value function.

There are many possible objectives under which the exploration and exploitation can be studied. One is the notion of *regret* — that is the loss of the algorithm compared to the execution of the *optimal* policy that knows the MDP. This objective does not generalize to discounted objective. When discounting is used, it is often impossible to make up for the initial exploration period. The alternative models therefore often focus on bounding the number of steps that are needed to stop exploring and start performing the policy (Kakade, 2003). This number of steps is known as *sample complexity.*

Typically, the goal of the theoretical analysis of the exploration–exploitation trade–off is to show that the sample complexity is polynomial in the relevant parameters of the MDP, such as $|\mathcal{S}|$, $|\mathcal{A}|$, $1/(1-\gamma)$, the error, and the confidence. Algorithms with polynomial sample complexity are known as *PAC-MDP*. A similar framework is KWIK (knows what it knows). This is similar to bounding the sample complexity, but does not require that the exploration steps happen initially; it, however, requires that the algorithm identifies when the step performed is an exploration step (Li, Littman, & Walsh, 2008).

In our work, we chose to use a simpler model for a few main reasons. First, many of the industrial problems that we are interested in have models available. It is then possible to gather the samples in an arbitrary fashion. This is because the samples are not a result of not knowing the environment; instead sampling is simply a means of reducing the computational complexity.

Second, the results on exploration and exploitation tradeoff have so far focused on problems with a sufficiently small number of states to be enumerated. Therefore, value function approximation — the main subject of this thesis — is not necessary. Nevertheless, the sampling bounds are quite loose and existing lower bound results are also pessimistic (Strehl, Li, & Littman, 2009). In particular, the lower bounds on the *sample complexity* depend linearly on $|\mathcal{S}| \cdot |\mathcal{A}|$ and probably quadratically on $1/(1-\gamma)$. Sample complexity is the number of steps after which the policy is close to optimal with high probability.

Another approach to bounding the sampling error is to not sample at all. Most sufficiently large domains are too large for all the states to be enumerated. Sometimes it is possible to reformulate the constraints so that it is sufficient to enumerate only a small subset of all states. These approaches require additional structure of the domain, typically a states space that is factored and transitions that are sparse (Guestrin et al., 2003). Appropriate structure that can be used to simplify sampling and solve practical problems is yet to be found.

Some previous work has analyzed the sampling error in value function approximation. Regularized iterative methods have been shown to converge under some restricted conditions (Farahmand, Ghavamzadeh, Szepesvari, & Mannor, 2009). These methods assume that the samples are gathered from executing the policy. In comparison with our work, these bounds are looser and the assumptions are stricter. In addition, the bounds cannot be computed online. However, the bounds apply to a more general setting since the samples cannot be gathered arbitrarily.

Sampling bounds have been also previously analyzed in for approximate linear programming (de Farias & van Roy, 2004). These bounds are however quite limited, because 1) they assume that the *optimal* state visitation frequency u^* is known, 2) they do not bound the policy loss but only $\|v^* - \tilde{v}\|_{1,c}$, 3) only bound state sampling error and ignore the transition-estimation error, and finally 4) require that the features contain Lyapunov features. In addition, some of our bounds are independent of the number of features — a useful property in feature selection proposed in Chapter 10.

9.7 Empirical Evaluation

This section demonstrates experimentally the approaches for estimating the state selection and transition estimation errors. We demonstrate that the methods work for the reservoir management problem, described Section B.3. We compare the approach to the local modeling assumption. This comparison is not completely fair, since the required parameters for the local modeling assumption are estimated from a supersample, but it gives a rough comparison.

The utility of a particular set of assumptions ultimately depends on whether it matches real problems. To evaluate our assumptions, we consider the reservoir management problem, described in Section B.3. This is a real problem estimated using historical data. We use it to evaluate whether the assumptions are realistic. The state embedding function in this case is identical as the function defined in Section B.3. The results of the comparison between the estimated values and true value of the model are shown in Figure 9.3.

For simplicity, we assumed a model that has a forecast of inflows and electricity prices of length 0. That is the states are identified only by the volume of water in the reservoir. The

	Features (ρ_1)		Rewards (ρ_2)	
	Mean	SD	Mean	SD
LM	19.54	14.23	27.41	12.2
GP	5.064	4.230	1.852	0.236

Figure 9.2. Comparison between the best-possible local modeling (LM) and a naive Gaussian process regression (GP) estimation of ϵ_p/ϵ_p^*. The results are an average over 50 random (non-uniform) sample selections. The table shows the mean and the standard deviation (SD).

simple state-action embedding function for this case is described in Section B.3. The state samples were for volumes 720000 to 1000000 in increments of 25000. There were 26 actions with discharges of 500 to 12000 in increments of 500. To evaluate the errors, and estimate the parameters for the local modeling assumption we used state samples for volumes 720000 to 1000000 in increments of 5000.

The Gaussian processes we used to approximate the functions ρ_1 and ρ_2 were identical. We use the squared exponential covariance defined as:

$$\kappa(s_1, s_2) = \exp\left(\frac{\|k(s_1) - k(s_2)\|}{2}\right)$$

These values were chosen a priori and were not optimized based on the sample. We assume that the domain of ρ_1 is $\mathbb{R}^3$ such that actions and features are mapped to a 3 dimensional real space. The actions and features are mapped linearly to the real space from the discrete values described in Section B.3. The volume was mapped by $k(\mathcal{S})$ to interval $[0, 10]$. Since there are many more acceptable values of water level than actions or features, the covariance between the water levels was assumed to be much stronger.

Figure 9.2 compares the 0.997 confidence interval of the Gaussian process regression with the local model estimation as a multiple of the true error ϵ_p^*. Note that the maximum of a confidence over multiple values of the Gaussian process is not the same as the confidence of the set. To evaluate the Gaussian process regression, we assumed the Matern covariance function with independent normal noise and computed the hyper-parameters from the samples. This is a standard approach used in Gaussian regression (Rasmussen & Williams, 2006); it does not preserve all the theoretical guarantees, but it allows to derive bounds from samples alone and prevents manual overfitting. In particular, the true error was overestimated by the Gaussian process in 85% of feature samples and 100% of reward samples. The constants K_ϕ, K_p, K_r in the local modeling assumption and the true error ϵ_p were estimated from a 5 to 1 supersample. These results indicate that the Gaussian process regression leads to tighter bounds than the local modeling assumption, even when the latter is estimated from data that is usually not available.

To evaluate the application of common random numbers, we used the reservoir management problem, defined in Section B.3. Figure 9.5 shows the uncertainty in estimating the value ρ_1 based on samples. It compares the results of using common random number versus using independent sampling. The grey regions represent the span of the functions for 5 independent runs, each assuming 5 samples per state.

We also analyzed the transition estimation error as a function of the number of sampled states. For this purpose, we also assume 5 samples per each state. The sampling load

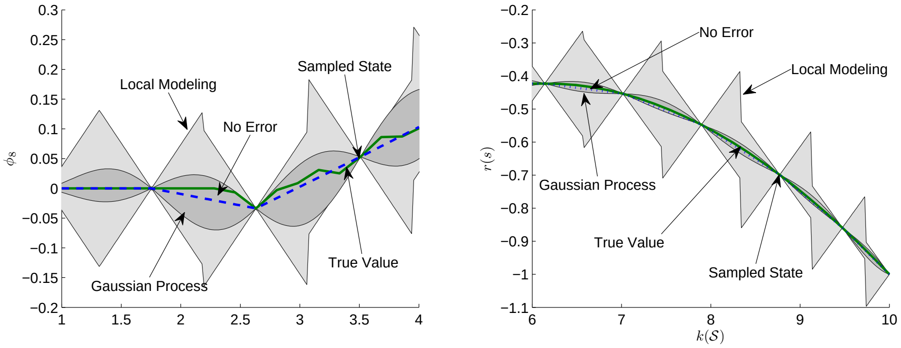

Figure 9.3. Comparison between the local modeling assumption and the Gaussian process regression for the Bellman residual for feature ϕ_8 and action a_5 (function ρ_1) and the reward for action a_1 (function ρ_2). Only a subset of $k(\mathcal{S})$ is shown to illustrate the detail. The shaded regions represent possible regions of uncertainties.

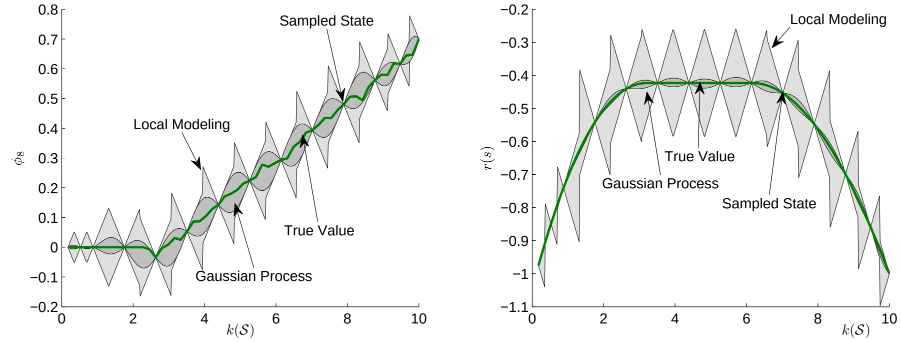

Figure 9.4. Comparison between the local modeling assumption and the Gaussian process regression for the Bellman residual for feature ϕ_8 and action a_5 (function ρ_1) and the reward for action a_1 (function ρ). The shaded regions represent possible regions of uncertainties.

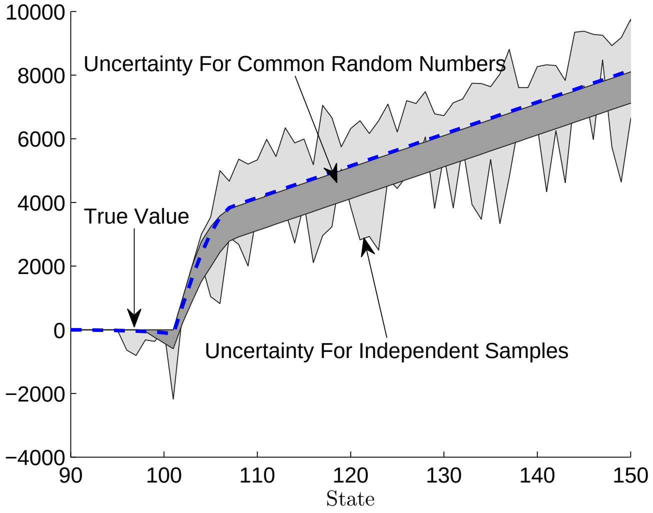

Figure 9.5. Uncertainty of the samples in reservoir management for the independent samples and common random numbers.

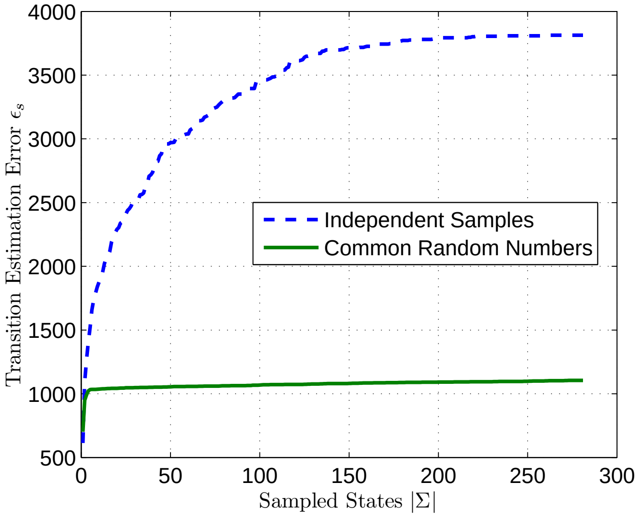

Figure 9.6. Transition estimation error as a function of the number of sampled states. The results are an average over 100 permutations of the state order.

for common random number is then typically lower, since sampling from the uncertainty model is often computationally difficult. We assume that the true model consists of 200 samples per state using common random numbers. It was not possible to compute the true values analytically. The results in Figure 9.6 show that the transition estimation error for independently–sampled constraints increases logarithmically, for the samples that use common random numbers, the error flattens after reaching the complexity of the random response function. This behavior is explained by Theorem 9.17.

The results of sampling on the blood inventory management problem were very similar. In fact, approximate linear programming did not lead to useful results even with a large number of samples if they were not sampled using common random numbers. The results reported in Chapter 4 therefore rely on common random number samples.

9.8 Contributions

This chapter presents a new method for estimating the sampling error in optimization–based algorithms. Virtually all the results presented in this chapter are new. These new sampling bounds are easier to compute than existing bounds for iterative algorithms. In particular, the bounds can utilize existing algorithms for function regression. It is important to note that our bounds apply directly to the policy loss; previous bounds for approximate linear programming only apply to auxiliary measures.

CHAPTER 10

FEATURE SELECTION

The majority of value function approximation methods assume that a *small* set of useful features is provided with the problem. These features must capture the essential properties of the problem in order to guarantee good solutions. It is desirable to lift this requirement and discover the relevant features automatically. This is, unfortunately, impossible in general, as the No Free Lunch theorem for machine learning states (Wolpert, 1996; Bousquet et al., 2004). Without features that are specified in advance, it is impossible to generalize beyond the current sample.

While features must be provided, it is important to focus on convenient ways of providing them. One option is to relax the requirement that the number of features is small. The possibility of using very rich feature spaces makes is significantly easier to capture the properties of the value function. In this chapter, we focus on methods that can use very large numbers of features. We call this approach "feature selection" though our focus in not on actually selecting features, just on using large feature spaces.

There are two main difficulties with using large number of features, which have been already addressed in the previous chapters. First, solving value function approximation with many features may be hard to solve. The feature selection methods rely on homotopy methods proposed in Chapters 6 and 7 for regularized approximate linear and bilinear programs. Second, solving value function approximation with many features may compromise generalization to unseen samples. The features selection methods therefore use the sampling bounds from Chapter 9.

The remainder of the chapter is organized as follows. First, Section 10.1 overviews the basic considerations in selecting features. Section 10.2 describes piecewise linear features, which are suitable for some of the application domains that we consider. Then, Section 10.3 presents an approach for feature selection for based on regularized approximate representation. This method relies strongly on the properties of optimization–based algorithms. Connections to related approaches for methods with a more flexibility in specifying features are discussed in Section 10.4, and finally, the approach is evaluated empirically in Section 10.5.

10.1 Feature Considerations

There are two main reasons for restricting the approximate value function v to be representable ($v \in \mathcal{M}$) : 1) it reduces the computational complexity of finding v, and 2) it helps to achieve generalization from a limited number of samples. This restriction is the source of the representational error — a fundamental component of the offline approximation error.

Usually, the simplicity and generality goals are satisfied by choosing $\mathcal{M}$ to be a small-dimensional linear space:

$$\mathcal{M} = \{\Phi x \,|\, x \in \mathbb{R}^m\},$$

where the matrix $\Phi : |\mathcal{S}| \times m$ has a "small" number of columns (e.g. $m = 50$). This feature-set makes it easy to solve the optimization problems in approximate linear and bilinear programs. It is, however, often hard to specify and the existing sampling bounds are not practically applicable.

For example, in some industrial applications, the features represent piecewise linear functions in a multi-dimensional continuous space, as discussed in Section 10.2. Features in this setting correspond to slope of the linear regions of the function. Designing a small number of features — that is linear regions — in a multidimensional space is difficult. The usual approach is, therefore, to treat each dimension of independently (Powell, 2007b). This may be interpreted as assuming that the utility of multiple resources is independent and may significantly degrade the solution quality.

A more flexible approach is to provide a large number of features and add a regularization function. Then:

$$\mathcal{M} = \{\Phi x \,|\, x \in \mathbb{R}^m, \ \|x\| \le \psi\}.$$

Here $\|\cdot\|$ is some norm. This choice of the representation can satisfy both requirements. As Chapter 9 shows, regularization ($\|x\| \le \psi$) can provide strong guarantees on the sampling error. Chapter 6 shows that when the regularization relies on the L_1 norm, the homotopy method may efficiently solve the problems (as long as the optimal solution is sparse).

The most common norm for regularization uses the L_2-norm. There are several advantages to using the L_1 norm in our setting. The L_1 norm preserves linearity of approximate linear and bilinear programs. In addition, it encourages solution sparsity which increases the efficiency of the homotopy methods and leads to simpler solutions. The experimental results show that the solutions are typically sparse; and we do not study this issue theoretically.

10.2 Piecewise Linear Features

This section overviews some based structures that can be used to define features and focuses specifically on piecewise linear feature, which are used in most of the experimental results. Piecewise linear functions must be used in some of the resource management domains — described in Section B.2 — to enable the computation of the greedy policy.

While the features may be arbitrary, it is often convenient to define them using functions in metric spaces. To do that, we use the *state embedding* function $k : \mathcal{S} \to \mathbb{R}^m$ (Definition 9.3). This function is also used in the sampling results in Chapter 9. Intuitively, it should map "similar" states to vectors that are close together. In many domains, this function naturally follows from the definition of the problem.

An example of a piecewise function in the blood inventory management problem — described in Section B.2 — is depicted in Figure 10.1. In this simple example, the state embedding function k maps states into $\mathbb{R}^2$ and can be defined as

$$k(s) = \begin{pmatrix} I_{AB} \\ I_0 \end{pmatrix},$$

where I_X represents the amount of blood of type X in the inventory.

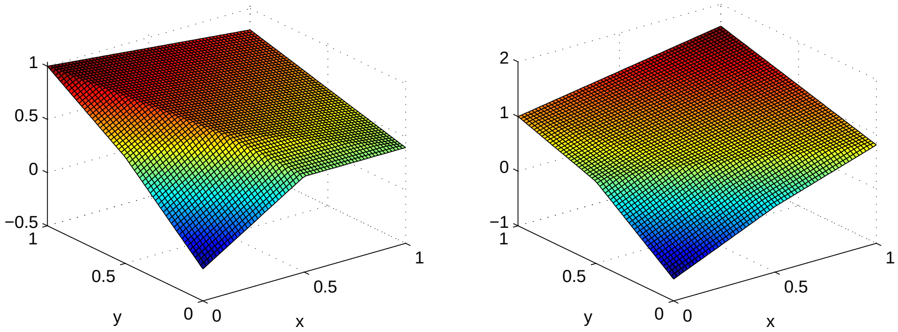

Figure 10.1. Examples of two piecewise linear value functions using the same set of 3 features. In blood inventory management, x and y may represent the amounts of blood in the inventory for each blood type.

The most straight-forward definition of a piecewise linear function is based on a triangulation of the $\mathbb{R}^n$. This is possible only when $k(\mathcal{S})$ is bounded, which is not a significant limitation in most applications. The value function for any states is then the linear interpolation of the values of the vertices of the corresponding polyhedron.

Because this is not a significant part of our approach, we only define these features only informally as follows. Let the triangulation be $T = \{T_1 \ldots T_l\}$ be a set of *non-overlapping convex* polyhedrons. The vertices of each polyhedron are defined as $T_i(1) \ldots T_i(n+1)$ — each of them is a vector. We use $x \in T$ to denote that a point is contained within polyhedron T. Then, the set of piecewise linear functions is defined as:

$$\phi_{ij}(s) = \begin{cases} y_j & k(s) \in T_j, \quad k(s) = \sum_{l=1}^{n+1} y_l T'(l) \\ 0 & \text{otherwise} \end{cases}$$

There is one feature for every vertex of the triangulation and the set of representable value functions is then simply a linear combination of these features. Note that the example in Figure 10.1 extends beyond the edges of the triangulation.

This definition has some significant limitations. First, the linear representation requires that the triangulation (that is the regions of linearity) is specified in advance. This is a problem limitation when the number of linear regions needs to be small in order to guarantee reasonably small sampling error. Second, the definition is cumbersome and it s hard to manipulate. Finally, it cannot be expected that if a value function is close to linear it could be faithfully represented using sparse coefficients. To address these issue we propose a different representation.

The piecewise linear basis system is defined as follows:

Definition 10.1 (Piece-wise Linear Basis System)**.** Assume a function $k : \mathcal{S} \to \mathbb{R}^n$ that maps states into a Euclidean space. The set of basis functions is defined for vectors $u_1, \ldots \in \mathbb{R}^n$ and scalars $t_{1,\ldots} \in \mathbb{R}$ as:

$$\phi_i(s) = \left[u_i^\mathsf{T} k(s) + t_i \right]_+ .$$

In addition, there is $\phi_0(s) = 1$.

The set of representable value functions is then defined as a linear combination of these features. This definition can also be easily used represent an infinite set of features.

The class of value functions that can be represented using the piecewise linear functions from Definition 10.1 is different than the class of functions representable using triangulation–based functions. We are not aware of deeper analysis of their approximation powers. An arbitrary number of features defined in this manner can be used easily with the regularized feature selection methods, proposed in Section 10.3.

It may often be necessary for the value function to be convex or concave. This is easy to ensure with piecewise linear functions, as the following proposition shows.

Proposition 10.2. *Let $v = \Phi x \in \mathcal{M}$ be piecewise linear as defined in Definition 10.1. Define a function $f(y) = v(s)$ when $y = k(s)$ on $k(\mathcal{S}) \subseteq \mathbb{R}^n$. When $x \geq \mathbf{0}$, the function f is convex and when $x \leq \mathbf{0}$ the function is concave. In addition, let I be a subset of features and $k(\mathcal{S}) \subseteq Z \subseteq \mathbb{R}^n$. Then the above holds for arbitrary values if x_i when $i \in I$ and $\phi_i(s)$ is linear on Z.*

The proof is straightforward using that fact that the function $[\cdot]_+$ is convex. Note that, for example, when $k(s) \geq 0$ for all $s \in \mathcal{S}$ the coefficient associated with the feature:

$$\phi(s) = \left[\mathbf{1}^\mathsf{T} k(s) + 0\right]_+$$

may be positive or negative without influencing the convexity or concavity of the function f. This is because $\phi(s)$ is linear on the set $Z = \{x \geq \mathbf{0}\}$.

10.3 Selecting Features

In this section, we present the main approach for solving approximate linear or bilinear programs with a large number of features. The ideas for solving linear and bilinear formulations are very similar; the main difference is that non-concavity of approximate bilinear programs complicates the derivation. Therefore, we first describe the approach for ALPs and then show how to generalize to ABPs.

The feature selection is based on regularized value functions. Even when value functions are regularized, this does not automatically guarantee a small sampling error. In particular, when the value of the regularization coefficient ψ is very large, the regularization may not play a significant role. Feature selection, therefore, entails selecting the appropriate coefficient ψ.

The choice of the regularization coefficient ψ has two main effects on the solution quality. First, with an increasing ψ also the set of representable value functions $\mathcal{M}(\psi)$ increases. That means that a larger set of value functions can be represented and the representational error decreases (or does not increase). Second, with an increasing ψ also the sampling error — represented by $\epsilon_c, \epsilon_p, \epsilon_s$ — increases, as Chapter 9 shows. The increase in the sampling error may often offset the decrease in the representational error. When the value of ψ decreases, the effects are reversed.

There is, therefore, no universally good value of ψ; it depends on the properties of the problem at hand. To solve reliably problems with many features, we propose methods that can compute ψ automatically based on the properties of the domain.

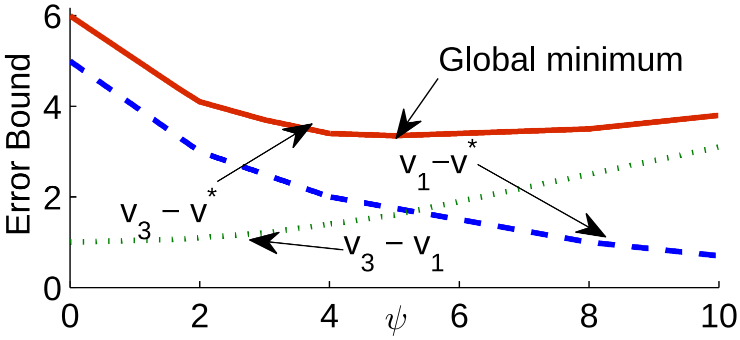

Figure 10.2. Sketch of error bounds as a function of the regularization coefficient. Here, v_1 is the value function of the full ALP, v_3 is the value function of the estimated ALP, and v^* is the optimal value function.

The main idea of our feature selection approach is to select ψ that balances off *bounds* on the sampling and representational errors. The true errors are to difficult (if not impossible) to compute to be used. This tradeoff is captured in Figure 10.2. The line $v_3 - v_1$ denotes the sampling error and the line $v_1 - v^*$ denotes the representational error. The red line is the bound on the total error — sum sampling and representational error. Our goal is find ψ that minimizes the bound on the total error — that is the global minimum.

Since we assume that the solutions are computed using homotopy methods and the function of the optimal solution is piecewise linear — as we discuss below — the minimum can be simply chosen from the solutions encountered during the optimization. The crucial question, however, is when to terminate the homotopy method. We show in the following how this is possible using convexity (or a similar property) of the error bounds in terms of ψ.

10.3.1 Approximate Linear Programming

For the ease of reference, the regularized *estimated* approximate linear program (ALP–R) is defined as follows:

$$\begin{aligned} \min_{x} \quad & \tilde{c}^{\mathsf{T}} \Phi x \\ \text{s.t.} \quad & \tilde{A} \Phi x \geq \tilde{b} \\ & \|x\|_{1,e} \leq \psi \end{aligned} \tag{e-ALP-R}$$

For most of the approach this section, is possible to use an arbitrary regularization norm (such as L_2 for example), but the L_1 norm provides the strongest guarantees.

The feature selection happens online using the homotopy algorithm to find a value of ψ that minimizes the comprehensive error bounds, such as Theorem 4.5. Note that it is irrelevant how the constants/functions $\epsilon_c(\psi), \epsilon_s(\psi), \epsilon_p(\psi)$ are computed, but Chapter 9 describes some approaches. The offline bounds in Theorem 4.4 could also be used, but we require that a bound on $\min_{v \in \mathcal{M}(\psi)} \|v - v^*\|_\infty$ is known, which is quite impractical.

The approach for selecting the optimal value of ψ is based on tracing the optimal objective of the approximate linear program as a function of the regularization coefficient ψ. This function is defined as follows.

Definition 10.3. The objective value of (e-ALP-R) with L_1 regularization as a function of ψ is denoted as $\theta_L(\psi)$.

A crucial property of the function θ_L is its convexity, as the following proposition shows.

Proposition 10.4. *The function $\theta_L(\psi)$ is convex and piecewise linear.*

The proof of the proposition can be found in Section C.11.

The convexity makes it possible to compute the optimal value ψ that minimizes the error bounds. The homotopy algorithm solves for the optimal value of the (e-ALP-R) for all values of the regularization coefficient ψ. To find the global minimum of the bounds, it is sufficient to use the homotopy method to trace $\theta_L(\psi)$ while its derivative is less than the sampling bound. The following corollary summarizes this.

Corollary 10.5. *Assume that $\epsilon_c(\psi)$, $\epsilon_p(\psi)$, and $\epsilon_s(\psi)$ are convex functions of ψ. Then, the error bound*

$$\|v(\psi) - v^*\|_{1,c} \leq f(\psi)$$

for an optimal solution $v(\psi)$ of (e-ALP-R) *is:*

$$f(\psi) = \theta_L(\psi) - c^\mathsf{T} v^* + \epsilon_c(\psi) + 2\frac{\epsilon_s(\psi) + \epsilon_p(\psi)}{1-\gamma}.$$

The function $f(\psi)$ is convex and its sub-differential[1] *$\nabla_\psi f$ is independent of v^*. Therefore, a global minimum ψ^* of f is attained when $0 \in \nabla_\psi f(\psi^*)$ or $\psi^* = 0$.*

The corollary follows directly from Proposition 10.4 and Theorem 4.5. In addition, the functions $\epsilon_s(\psi), \epsilon_p(\psi), \epsilon_c(\psi)$ are linear functions of ψ in bounds derived in Chapter 9.

10.3.2 Approximate Bilinear Programming

The application to approximate bilinear programs is more complex, since the corresponding function θ_B is not necessarily convex (see Proposition 7.3). For the ease of reference, the regularized *estimated* approximate bilinear program (ABP–L_∞) is defined as follows:

$$\begin{array}{llll}
\min\limits_{\pi \,|\, \lambda,\lambda',v} & \pi^\mathsf{T}\lambda + \lambda' & & \\
\text{s.t.} & \tilde{B}\pi = \mathbf{1} & \tilde{A}\Phi x - \tilde{b} \geq \mathbf{0} & \\
& \pi \geq \mathbf{0} & \lambda + \lambda'\mathbf{1} \geq \tilde{A}\Phi x - \tilde{b} & \qquad\text{(e-ABP–}L_\infty) \\
& & \lambda, \lambda' \geq \mathbf{0} & \\
& & \|x\|_{1,e} \leq \psi &
\end{array}$$

We study only the robust formulation, but the results for other formulations are very similar.

The optimal solution of (e-ABP–L_∞) is denoted as $\theta_B(\psi)$:

$$\theta_B(\psi) = \min_{v \in \mathcal{M}(\psi) \cap \mathcal{K}} \|v - Lv\|_\infty,$$

following Definition 7.2. This function may not be convex, as Proposition 7.3 shows. It is, however, piecewise linear.

[1] Function f may be non-differentiable

Proposition 10.6. *The function $\theta_B(\psi)$ is piecewise linear.*

Proof. The proof is straightforward using that: 1) the optimal objective function of a regularized linear program is piecewise linear (Proposition 6.5), and 2) $\theta_B(\psi) = \min_\pi f_\pi(\psi)$ where $f_\pi(\psi)$ is a linear function (Lemma 5.4). A minimum of a *finite* set of piecewise linear functions is also linear.

While θ_B may not be convex, it is possible to establish similar properties of the regularization functions as for ALPs. This makes it possible to determine the termination points for the homotopy method. However, the optimal value of ψ may not be in this terminal point. We use Theorem 7.4 and to establish the following.

Corollary 10.7. *Assume that $\epsilon_p(\psi)$, and $\epsilon_s(\psi)$ are convex functions of ψ and that for every state $s \in \mathcal{S}$ we have that $r(s, a_1) = r(s, a_2)$ for all $a_1, a_2 \in \mathcal{A}$. Then, the error bound*

$$\|v(\psi) - Lv(\psi)\|_\infty \leq f(\psi)$$

for an optimal solution $v(\psi)$ of (e-ABP–L_∞) *is:*

$$f(\psi) = \theta_B(\psi) + 2\epsilon_s(\psi) + \epsilon_p(\psi).$$

Let $\bar{\psi}$ be minimal such that:

$$\frac{\theta_B(0) - \theta_B(\bar{\psi})}{\bar{\psi}} \leq \frac{d}{d\psi}(\epsilon_p + 2\epsilon_s).$$

The global minimum ψ^* of f is attained on the interval $[0, \bar{\psi}]$ and is in one of the finite nonlinearity points of θ_B given that $\epsilon_p(\psi), \epsilon_s(\psi)$ are linear.

□

The result in Corollary 10.7 is much stronger than the feature selection method in Corollary 10.5. In particular, it does not contain the coefficient $1/(1-\gamma)$, which can be very large. Is also applies to $\|v - Lv\|_\infty$ which can be directly used to bound the policy loss (Theorem 2.16). It does not require that bounds on u_π are known. It is again, like in many other settings, much harder to solve.

10.4 Related Work

Regularization using the L_1 norm has been widely used in regression problems by methods such as LASSO (Hastie et al., 2009) and Dantzig selector (Candes & Tao, 2007). The value-function approximation setting is, however, quite different and the regression methods are not *directly* applicable. Regularization has been previously used in value function approximation (Farahmand et al., 2009; Kolter & Ng, 2009). In comparison with LARS-TD (Kolter & Ng, 2009), a regularized value function approximation method, we explicitly show the influence of regularization on the sampling error, provide a well-founded method for selecting the regularization parameter, and solve the full control problem.

The methods for more flexible feature definition are often called *basis construction.* The empirical work on basis construction is very encouraging. For example, Mahadevan (Mahadevan, 2005c) has recently reported significant improvements in solution quality of optimized basis selection compared with a handcrafted initial basis. The performance of this method on a simple problem however reveals that it sometimes constructs a poor basis that does not improve monotonically with the number of iterations.

The work on basis construction within the reinforcement learning community has produced mostly value solution techniques for problems with unknown models. Many of the proposed reinforcement learning algorithms have shown to produce promising experimental results (Mahadevan, Maggioni, Ferguson, & Osentoski, 2006; Johns & Mahadevan, 2007; Parr, Painter-Wakefield, Li, & Littman, 2007). Closely related work in operations research has addressed the problem of aggregation in linear programs.

For example, in recent work that is representative of several reinforcement learning efforts, Mahadevan has used spectral analysis of the discrete state space to build the approximation basis (Mahadevan, 2005a, 2005b, 2005c). The method is also known as *representation policy iteration* (RPI). These bases can be shown to provide a good approximation for functions that are "smooth" on a given manifold in terms of a *Sobolev norm*, a generalization of L_2 norm. The basis is constructed using the *top eigenvectors of the Laplacian* of the transition graph. The transition graph is constructed by simulating a *random policy* and connecting states whenever there is a transition between them. RPI has been extended to problems with continuous state spaces (Mahadevan & Maggioni, 2005), where the transition graph is constructed using a k-nearest-neighbors method to determine the samples that represent each of its nodes. The resulting approximate model can be solved using LSPI (Lagoudakis & Parr, 2003).

While the approach has been applied successfully to various artificial domains, it suffers from some deficiencies. First, the value function may not be smooth with regard to the constructed graph (Petrik, 2007). The precise conditions that guarantee smoothness have not yet been formalized. Moreover, the use of the transition graph is only weakly motivated and does not lead to any performance guarantees. One concern is that when the graph is created, the direction and probability of the transitions are ignored. The lack of directionality was partially remedied in (Johns & Mahadevan, 2007), but without any bounds on the resulting performance. Finally, the approach does not account for the fact that there may be multiple policies for the input problem. Therefore, even if convergence can be proved, it would most likely be limited to problems with a single policy. A related value method uses *neighborhood component analysis* (NCA) to create the basis functions (Keller, Manor, & Precup, 2006; Bertsekas & Castalion, 1989).

An alternative approach for continuous problems, commonly used in operations research, is to discretize the state-space and estimate the transition matrix. This leads to error bounds based on the continuity of the value function and the discretization used (Rust, 1997). The difficulty with this method, however, is that the error bounds are based on the optimal value function that is not known. Iterative improvement of the basis for a similar problem was proposed by Trick and Stanley (1993) and Trick and Zin (1997). However, the convergence of this method has not yet been theoretically studied.

Most of the previously described methods rely on global analysis of the state space. A simpler alternative is to solve the problem with an arbitrary basis and then improve the basis based on the obtained solution. One option is to add the Bellman residual of the obtained value function to the basis (Parr et al., 2007). This method was shown to converge for problems with a single policy, but only at the rate of value iteration, which is slow for discount factors close to 1. Our analysis shows that the basis obtained in this case is the same as the *Krylov basis* (Petrik, 2007). The inherent problem with the approach is that it is not guaranteed to converge for problems with multiple policies. A similar approach was proposed for factored MDPs, when the one-step transition function can be represented as a Bayesian network (Guestrin et al., 2003). Approximate linear programming is attractive in this case, because the number of constraints can be significantly reduced when the structure of the problem is considered and a specific basis is assumed. A method for iteratively adding new basis functions — including multiplier-update type vectors — was proposed in (Patrascu et al., 2002; Patrascu, 2004; Poupart et al., 2002).

In related work in operations research, aggregation is often used to reduce problem size in linear programs (Litvinchev & Tsurkov, 2003; Rogers, Plante, Wong, & Evans, 1991). It is often used when the available data is insufficient for building a precise model (Zipkin, 1977). The two main types of aggregation are variable aggregation, and constraint aggregation, in which both variables and constraints are aggregated. Unlike aggregation in MDPs, aggregation in linear programs is often weighted. Thus, approximate linear programming, as introduced above, is a form of variable aggregation. In addition to aggregation, research efforts in this area also seek various methods of *disaggregation*, that is, obtaining a solution in the original space. Most of this research has focused on developing a priori and a posteriori error bounds (Litvinchev & Tsurkov, 2003; Mendelssohn, 1980; Shetty & Taylor, 1987; Zipkin, 1977, 1978). These bounds are in general tighter than the existing approximate linear programming bounds. However, they often rely on assumptions that make them impractical for MDPs.

Vakhutinsky, Dudkin, and Ryvkin (1979) have developed an iterative approximation algorithm for an input-output model formulated as a linear program. The algorithm is formulated for general linear programs, and is based on *variable* aggregation and optimal disaggregation. Only local convergence is guaranteed. A general framework for iterative variable aggregation and disaggregation is developed in (Litvinchev & Tsurkov, 2003) (Chapter 2). It is based on gradient descent on the optimization basis with regard to the current solution. Its drawback is that the number of constraints is as large as in the original problem and convergence is not proved.

An interesting iterative aggregation-disaggregation algorithm for solving MDPs has been developed using *both* variable and constraint aggregation (Mendelssohn, 1982). The algorithm uses optimal disaggregation (Zipkin, 1978) and multiplier-method type update of the dual variables. However, the convergence properties of this method need to be further investigated.

10.5 Empirical Evaluation

We demonstrate and analyze the properties of feature selection for ALP on a simple chain problem with 200 states, in which the transitions move to the right by one step with a centered Gaussian noise with standard deviation 3. The reward for reaching the rightmost state was +1 and the reward in the 20th state was -3. This problem is small to enable calculation of the optimal value function and to control sampling. We uniformly selected every fourth state on the chain and estimated the sampling bound $\epsilon_p(\psi) = 0.05\psi$. The approximation basis in this problem is represented by piecewise linear features, of the form $\phi(s_i) = [i - c]_+$, for c from 1 to 200; these features were chosen due to their strong guarantees for the sampling bounds. The experimental results were obtained using the proposed homotopy algorithm.

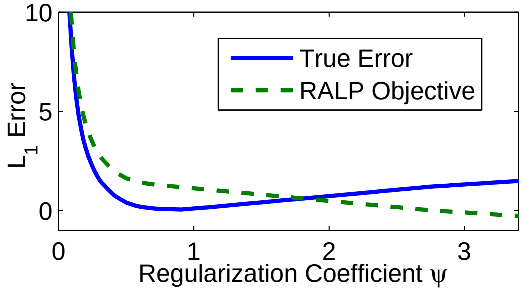

Figure 10.3. Comparison of the objective value of regularized ALP with the true error.

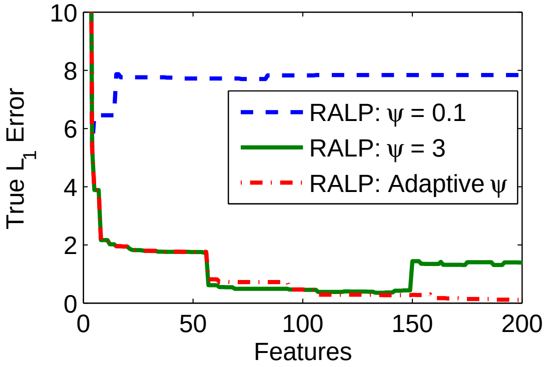

Figure 10.4. Comparison of the performance regularized ALP with two values of ψ and the one chosen adaptively (Corollary 10.5).

Figure 10.3 demonstrates the solution quality of (e-ALP-R) on the chain problem as a function of the regularization coefficient ψ. The figure shows that although the objective of (e-ALP-R) keeps decreasing as ψ increases, the sampling error overtakes that reduction. It is clear that a proper selection of ψ improves the quality of the resulting approximation. To demonstrate the benefits of regularization as it relates to overfitting, we compare the performance of ALP with and without regularization as a function of the number of available features in Figure 10.5. While ALP performance improves initially, it degrades severely with more features. The value ψ in the regularized ALP is selected automatically using Corollary 10.5 and the sampling bound of $\epsilon_p(\psi) = 0.5\psi$. Figure 10.4 demonstrates that regularized ALP may also overfit, or perform poorly when the regularization coefficient ψ is not selected properly. To find the proper value of ψ, as described in Corollary 10.5, the problem needs to be solved using the homotopy method described in Chapter 6.

10.6 Contributions

The main contribution described in this chapter is the feature selection method. While this is only a short part of the chapter, the result mostly summarizes results from other chapters. The other minor contribution of this chapter is the new definition of piecewise

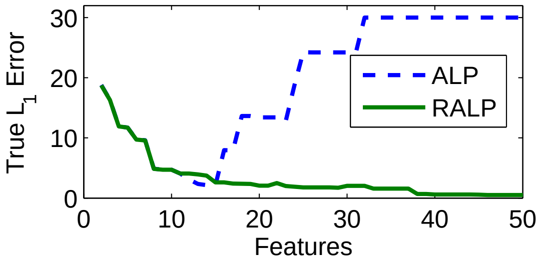

Figure 10.5. Average of 45 runs of ALP and regularized ALP as a function of the number of features. Coefficient ψ was selected using Corollary 10.5.

linear functions. The piecewise linear functions can represent functions that are close to linear using sparse coefficients; this makes them convenient to use with the L_1 regularization.

CHAPTER 11

HEURISTIC SEARCH

This chapter draws a connection between the work described in this thesis — mostly in the context of reinforcement learning domains — and work on automated planning. Like the reinforcement learning community, the planning community also aims to solve sequential decision processes, but with a somewhat different considerations. While value function approximation is most often used to solve long-horizon maintenance problems — such as ones involved in managing resources — planning domains are typically about achieving a goal in shortest number of steps. In addition, the models of planning domains are often much richer than plain MDPs and this structure aids in developing solution methods.

There are many similarities between approximate dynamic programming and methods used to solve planning problems. For example, the planning community relies on a heuristic function, which also assigns a value to each state just like a value function. The main difference is in how the heuristic function is used. Value functions in ADP are typically used to construct a greedy policy, while the planning community relies on more sophisticated methods, such as heuristic search. This chapter contrasts the approximate dynamic programming methods with heuristic search in more detail. We also show in this chapter how approximate linear programming — an optimization–based method — can be used to solve planning problems.

The chapter intentionally stands apart from the remainder of the thesis to make it easier to understand the connections without requiring a deep understanding of optimization–based value function approximation.

11.1 Introduction

An important challenge in planning research is to develop *general-purpose* planning systems that rely on *minimal human guidance.* General-purpose planners — also referred to as *domain independent* — can be easily applied to a large class of problems, without relying on domain-specific assumptions. Such planners often take as input problems specified using PDDL (Gerevini & Long, 2005), a logic-based specification language inspired by the well-known STRIPS formulation (Fikes & Nilsson, 1971; Russell & Norvig, 2003).

Planning plays an important role in many research areas such as robotics, reinforcement learning, and operations research. Examples of planning problems in the various research fields are:

Mission planning — How to guide an autonomous spacecraft? These problems are often modeled using highly-structured and discrete state and action spaces. They have been addressed extensively by the classical planning community (Ghallab, Nau, & Traverso, 2004).

Inventory management — What is the best amount of goods to keep in stock to minimize holding costs and shortages? These problems are often modeled using large continuous state spaces and have been widely studied within operations research (Powell, 2007a).

Robot control — How to control complex robot manipulations? These stochastic planning problems have a shallow state-space structure often with a small number of actions, and have been studied by the reinforcement learning community (Sutton & Barto, 1998).

Interestingly, despite the diverse objectives in these research areas, many of the solution methods are based on heuristic search (Ghallab et al., 2004). However, different research fields sometimes emphasize additional aspects of the problem that we do not address here. Reinforcement learning, for example, considers model learning to be an integral part of the planning process.

Planning problems can be solved in several different ways. One approach is to design a *domain-specific* solution method. Domain-specific methods are often very efficient but require significant effort to design. Another approach is to formulate the planning problem as a *generic optimization* problem, such as SAT, and solve it using general-purpose SAT solvers (Kautz & Selman, 1996). This approach requires little effort, but often does not exploit the problem structure. Yet another approach, which is the focus of this chapter, is to use *heuristic search* algorithms such as A* or branch-and-bound (Bonet & Geffner, 2001; Russell & Norvig, 2003). Heuristic search algorithms represent a compromise between domain specific methods and a generic optimization. The domain properties are captured by a heuristic function which assigns an approximate utility to each state. Designing a heuristic function is usually easier than designing a domain-specific solver. The heuristic function can also reliably capture domain properties, and therefore heuristic search algorithms can be more efficient than generic optimization methods.

The efficiency of heuristic search depends largely on the accuracy of the heuristic function, which is often specifically designed for each planning problem. Designing a good heuristic function, while easier than designing a complete solver, often requires considerable effort and domain insight. Therefore, a truly autonomous planning system should not rely on hand-crafted heuristic functions. To reduce the need for constructing heuristics manually, much of the research in autonomous planning concentrates on automating this process. The goal of automatic heuristic construction is to increase the applicability of heuristic search to new domains, not necessarily to improve the available heuristics in well-known domains.

The construction of heuristic functions often relies on *learning* from previous experience. While constructing a good heuristic function can be useful when solving a single problem, it is especially beneficial when a number of related problems must be solved. In that case, early search problems provide the necessary samples for the learning algorithm, and the time spent on constructing a good heuristic can be amortized over future search problems.

The focus of this chapter is on constructing heuristic functions within a planning framework. The specific context of construction within the planning system is also crucial in designing the learning algorithms. Planning, despite being similar to generic optimization problems, such as SAT, usually encompasses a number of other components. A general framework for learning heuristics in planning systems is shown in Figure 11.1. The ovals represent inputs, the rectangles represent computational components, and the arrows represent information

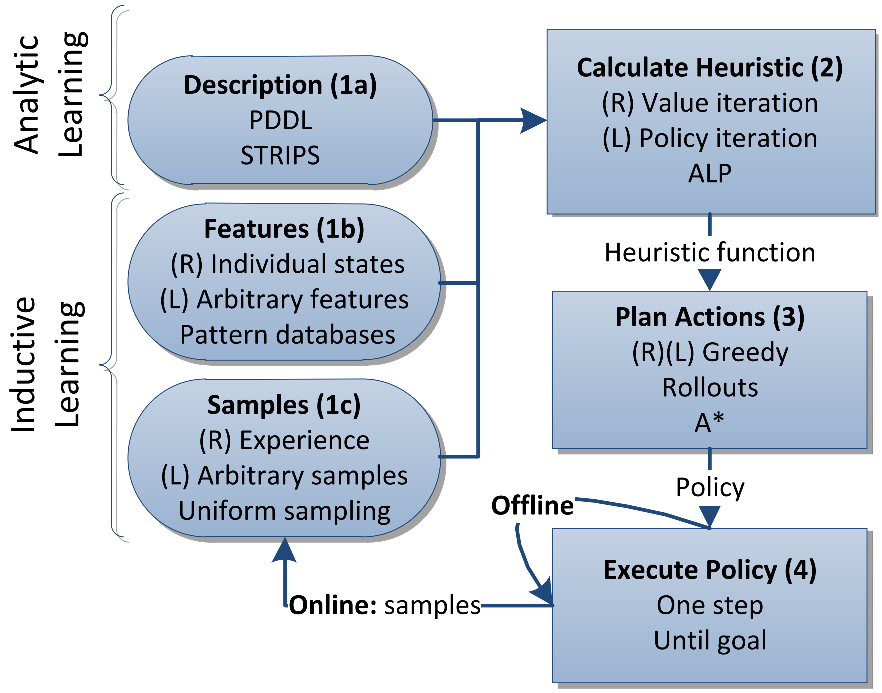

Figure 11.1. Framework for learning heuristic functions. The numbers in parentheses are used to reference the individual components.

flow. The computational components list sample implementations and the inputs list sample information sources.

The inputs are formally defined in Section 11.3. Intuitively, the problem description (1a) denotes a precise model in some description language that can be easily manipulated. The description languages are often logic-based, such as STRIPS or PDDL. The features (1b) represent functions that assign a set of real values to each state. Finally, the samples (1c) represent simple sequence of states and actions that respect the transition model.

The methods for constructing heuristics can be divided into two broad categories based on the input they require: 1) analytic methods and 2) inductive methods (Zimmerman & Kambhampati, 2003). *Analytic methods* use the formal description of the domain in order to derive a relaxed version of the problem, which in turn is used to derive a heuristic. For example, an integer linear program may be relaxed to a linear program (Bylander, 1997), which is much easier to solve. A STRIPS problem may be abstracted by simply ignoring selected literals (Yoon, Fern, & Givan, 2008). When applicable, analytic methods work well, but the required description is sometimes unavailable.

Inductive methods rely on a provided set of features and transition samples of the domain. A heuristic function is calculated as a function of these *state features* based on *transition samples*. Inductive methods are easier to apply than analytic ones, since they rely on fewer assumptions. Moreover, the suitable structure needed to use abstraction is rarely present in stochastic domains (Beliaeva & Zilberstein, 2005).

Inductive heuristic learning methods can be further classified by the source of the samples into *online* and *offline* methods. Online methods interleave execution of a calculated plan with sample gathering. As a result, a new plan may be often recalculated during plan execution. Offline methods use a fixed number of samples gathered earlier, prior to plan execution. They are simpler to analyze and implement than online methods, but may perform worse due to fewer available samples.

To illustrate the planning framework, consider two common planning techniques that can learn heuristic functions: Real-Time Dynamic Programming (RTDP) (Barto, Bradtke, & Singh, 1995) and Least Squares Policy Iteration (LSPI) (Lagoudakis & Parr, 2003). Both methods are examples of inductive learning systems and do not rely on domain descriptions. The component implementations of RTDP and LSPI are marked with "(R)" and "(L)" respectively in Figure 11.1. The methods are described in more detail in Section 11.3.3.

RTDP is an online method that is often used to solve stochastic planning problems. While it is usually considered a type of heuristic search, it can also be seen as an inductive method for learning heuristic functions. Because features are independent for each state, it is possible to represent an arbitrary heuristic function as shown in Section 11.3.2. However, RTDP updates the heuristic function after obtaining each new sample during execution, thus it does not generalize to unseen states.

LSPI is an offline method that uses a fixed set of predetermined samples. It has been developed by the reinforcement learning community, which often refers to the heuristic function as the *value function*. The features may be arbitrary and therefore could limit the representable heuristic functions. However, unlike RTDP, LSPI also generalizes value to states that have not been visited.

Figure 11.1 represents a single planning system, but it is possible to combine multiple learning systems. Instead of directly using the calculated heuristic function to create a policy, the heuristic function can be used as an additional feature within another planning system. This is convenient, because it allows analytic methods to create good features to be used with inductive learning.

The remainder of the chapter presents a unified framework for learning heuristic functions. In particular, it focuses on calculating heuristics — component (2) of the general framework — but it also describes other components and their interactions. Section 11.2 introduces the framework and the search algorithms used in components (3) and (4). Section 11.3 describes the existing inductive and analytic approaches to learning heuristic functions. The chapter focuses on inductive learning methods — the analytic methods are described because they can serve as a good source of features. Section 11.4 describes an inductive method for constructing heuristic functions using linear programming. The linear programming method is further analyzed in Section 11.5. Experimental results are presented in Section 11.6.

This chapter concentrates on using previous experience to create or improve heuristic functions. Previous experience can be used in a variety of ways. For example, some planners use samples to learn control rules that speed up the computation (Ghallab et al., 2004). Early examples of such systems are SOAR (Laird, Rosenbloom, & Newell, 1986) and PRODIGY (Minton, Knoblock, Kuokka, Gil, Joseph, & Carbonell, 1989). Other notable approaches include learning state invariants (Rintanen, 2000), abstracting planning rules (Sacerdott, 1974), and creating new planning rules (Leckie & Zuckerman, 1998). In comparison, learning a heuristic function offers a greater flexibility, but it is only relevant to planning systems that use heuristic search.

11.2 Search Framework

This section formally defines the search framework and describes common search algorithms. The section also discusses metrics of heuristic-function quality that can be used to measure the required search effort. The search problem is defined as follows:

Definition 11.1. A search problem is a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ is a *finite* set of states with a single initial state σ and a single goal state τ. $\mathcal{A}$ is a *finite* set of actions available in each state, and the function $P(s, a) = s'$ specifies the successor state s' of some state $s \neq \tau$ when action a is taken. The set *finite* of all successors of s is denoted as $\mathcal{C}(s)$. The function $r(s, a) \in \mathbb{R}$ specifies the reward. A discount factor $0 < \gamma \leq 1$ applies to future rewards.

The discount factor γ is considered only for the sake of generality; the framework also applies to search problems without discounting (i.e. $\gamma = 1$). The assumption of a single goal or initial state is for notational convenience only. Problems with many goals or initial states can be easily mapped into this framework.

The solution of a search problem is a *policy* $\pi : \mathcal{S} \to \mathcal{A} \cup \{\nu\}$, which defines the action to take in each state. Here, ν represents an undefined action and by definition $\pi(\tau) = \nu$. The policy is chosen from the set of all valid policies Π. In a deterministic search problem, a policy defines a *path* from a state s_1 to the goal. The path can be represented using the following sequence:

$$(\{s_i \in \mathcal{S}, a_i \in \mathcal{A} \cup \{\nu\}\})_{i=1}^{n},$$

such that for $i < n$:

$$\begin{aligned} a_i &\in \mathcal{A} & a_i &= \pi(s_i) \\ t(s_i, a_i) &= s_{i+1} & s_n &= \tau \end{aligned}$$

The *return*, or value, of such a sequence for a given policy π is:

$$v_\pi(s_1) = \sum_{i=1}^{n-1} \gamma^i r(s_i, a_i).$$

The optimal value, denoted as $v^*(s_1)$, is defined as:

$$v^*(s) = \max_{\pi \in \Pi} v_\pi(s).$$

The optimal value of a state is the maximal possible value over any goal-terminated sequence starting in that state. The objective is to find a policy π^* that defines a path for the initial state σ with the maximal *discounted cumulative* return $v_{\pi^*}(\sigma)$.

Notice that the optimal policy only needs to define actions for the states connecting the initial state σ to the goal state τ. For all other states the policy may have the value ν. For simplicity, assume that there is always a path from σ to any state s and that there is a path from any state s to the goal τ. It is easy to modify the techniques for problems with dead ends and unreachable states. To guarantee the finiteness of the returns when $\gamma = 1$, assume that there are no cycles with positive cumulative rewards.

Algorithm 11.1: A* Heuristic Search Algorithm

```
Input: Heuristic function $v$, initial state $\sigma$
$\mathcal{O} \leftarrow s_0$ ;                                   // Initialize the open set
$s' \leftarrow \arg\max_{s \in \mathcal{O}} f(s)$ ;                       // $f(s) = g(s) + v(s)$
while $s' \neq \tau$ do
    $\mathcal{O} \leftarrow \mathcal{O} \setminus \{s'\}$ ;                      // Remove state $s'$ from the open set
    $\mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{C}(s')$ ;              // Append the one with the greatest $g$ value
    $s' \leftarrow \arg\max_{s \in \mathcal{O}} f(s)$ ;                   // $f(s) = g(s) + v(s)$
```

11.2.1 Heuristic Search

Heuristic methods calculate a policy for search problems from a heuristic function and an initial state. A *heuristic function* $v : \mathcal{S} \to \mathbb{R}$ returns for each state s an estimate of its optimal value $v^*(s)$. An important property of a heuristic function is admissibility, which guarantees optimality of some common algorithms. A heuristic function v is admissible when

$$v(s) \geq v^*(s) \text{ for all } s \in \mathcal{S}.$$

Notice we consider maximization problems instead of more traditional minimization problems to be consistent with the reinforcement learning literature. Therefore, an admissible heuristic is an *overestimate* of the optimal return in our case.

Admissibility is implied by *consistency*, which is defined as:

$$v(s) \geq r(s,a) + \gamma v(P(s,a)) \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}.$$

Consistency is often necessary for some heuristic search algorithm to be efficient.

The simplest heuristic search algorithm is *greedy best-first search*. This search simply returns the following policy:

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \{r(s,a) + \gamma v(s)\},$$

with ties broken arbitrarily. Greedy search does not require the initial state and defines a valid action for each state.

Many contemporary heuristic search methods are based on the A* algorithm, a simplified version of which is depicted in algorithm 11.1. It is a best-first search, guided by a function $f : \mathcal{S} \to \mathbb{R}$, defined as:

$$f(s) = g(s) + v(s).$$

The total reward (possibly negative) achieved in reaching s from the initial state σ is denoted as $g(s)$. States are chosen from an open set $\mathcal{O}$ according to their f value. The successors of the chosen states are consequently added to the open set. A* is guaranteed to find the optimal solution when the heuristic function is *admissible*.

A* is an optimally efficient search algorithm since it expands the minimal number of states needed to prove the optimality of the solution with a *consistent* heuristic function. When the heuristic function is inconsistent, the efficiency of A* could be improved using simple modifications such as PATHMAX (Zhang, Sturtevant, Holte, Schaeffer, & Felner, 2009).

Despite the theoretical optimality of A*, its applicability is often limited by the exponential growth of the open list $\mathcal{O}$. A* modifications have been proposed that address its large memory requirements. One common modification is Iterative Deepening A* (IDA*), which repeatedly searches the graph in a depth-first manner with a decreasing threshold on the value f of states (Korf, 1985). Others have proposed modifications that do not guarantee optimality, such as weighted A* (Pohl, 1970; Hansen & Zhou, 2007; Thayer & Ruml, 2008). Because most of these algorithms retain the basic characteristics of A* and use the same heuristic function, the rest of the chapter focuses on A* as a general representative of these heuristic search methods.

The presented framework also applies to stochastic domains, formulated as Markov decision processes (MDP) (Puterman, 2005). An MDP is a generalization of a search problem, in which the successor state is chosen stochastically after taking an action. A* is not directly applicable in stochastic domain, but some modification have been proposed. The most commonly used heuristic search algorithm for MDPs is the greedy best-first search method described above. More sophisticated algorithms used in stochastic domains are RTDP and LAO*, which are discussed in Section 11.3.2. However, in some domains, even greedy search can be overly complicated (Powell, 2007a).

11.2.2 Metrics of Heuristic Quality and Search Complexity

This section discusses the influence of a heuristic function quality on the search complexity. To start, it is necessary to define a measure of the heuristic quality. The measure must accurately capture the time complexity of the search process and also must be easy to estimate and optimize. For example, the number of states that A* expands is a very precise measure but is hard to calculate. As it turns out, a precise and simple quality measure does not exist yet, but approximations can be used with good results.

The discussion in this section is restricted to admissible heuristic functions. This is a limitation when a suboptimal solution is sought and admissibility is not crucial. However, the analysis of inadmissible heuristics is hard and involves trading off solution quality with time complexity. Moreover, admissible heuristic functions can be used with suboptimal algorithms, such as weighted A* (Pohl, 1970), to obtain approximate solutions faster.

Early analysis shows bounds on the number of expanded nodes in terms of a worst-case additive error (Pohl, 1977; Gaschnig, 1979):

$$\epsilon = \max_{s \in \mathcal{S}} |v(s) - v^*(s)|.$$

The bounds generally show that even for a small value of ϵ, a very large number of nodes may be expanded unnecessarily. A more general analysis with regard to errors weighted by the heuristic function shows similar results (Pearl, 1984). In addition, most of these bounds assume a single optimal solution, which is often violated in practice.

Recent work shows tight bounds on the number of nodes expanded by A* in problems with multiple solutions (Dinh, Russell, & Su, 2007). In particular, the number of expanded nodes may be bounded in term of

$$\epsilon = \max_{s \in \mathcal{S}} \frac{|v(s) - v^*(s)|}{|v^*(s)|}.$$

This work assumes that there is a relatively small number of optimal solutions clustered together.

The existing bounds usually require a small number of goal states and states that are close to them. These assumptions have been questioned because many benchmark problems do no satisfy them (Helmert & Roger, 2008; Helmert & Mattmuller, 2008). When the assumptions do not hold, even a good heuristic function according to the measures may lead A* to explore exponentially many states. In light of this evidence, the utility of the existing quality bounds is questionable.

Because a widely acceptable measure of the quality of heuristic functions does not yet exist, we focus on two objectives that have been studied in the literature and are relatively easy to calculate. They are:

L_1**, the average–case error:** $\|v - v^*\|_1 = \sum_{s\in\mathcal{S}} |v(s) - v^*(s)|$, and
L_∞**, the worst–case error:** $\|v - v^*\|_\infty = \max_{s\in\mathcal{S}} |v(s) - v^*(s)|$.

11.3 Learning Heuristic Functions

This section overviews existing methods for learning heuristic functions in planning domains. Most of these methods, however, can also be applied to search. As mentioned earlier, methods that use experience to learn control rules or other structures are beyond the scope of this chapter (Zimmerman & Kambhampati, 2003). First, to classify the methods based on the inputs in Figure 11.1, we define the inputs more formally below.

Description (1a) The domain description defines the structure of the transitions and rewards using a well-defined language. This description allows to easily simulate the domain and to thoroughly analyze its properties. Domain descriptions in planning usually come in one of three main forms (Ghallab et al., 2004): 1) *Set-theoretic representation* such as STRIPS, 2) *classical representation* such as PDDL, or 3) *state-variable representation* such as SAS+. Extensive description of these representations is beyond the scope of this chapter. The details can be found in (Ghallab et al., 2004).

Features (1b) A feature is a function $\phi : \mathcal{S} \to \mathbb{R}$ that maps states to real values. One is typically given a set of features:

$$\Phi = \{\phi_1 \ldots \phi_k\}.$$

The heuristic function is then constructed using a combination function $\bar{\theta}$:

$$v(s) = \bar{\theta}(\phi_1(s), \phi_2(s), \ldots, \phi_k(s)). \tag{11.1}$$

The space of combination functions Θ is often restricted to linear functions because of their simplicity. Using linear θ, the heuristic function is expressed as:

$$v(s) = \theta(\phi_1(s), \phi_2(s), \ldots, \phi_k(s)) = \sum_{i=1}^{k} x_i \phi_i(s).$$

The actual combination function $\bar{\theta}$ may be either fixed for the domain or may be computed from other inputs.

One simple way to obtain features is state aggregation. In aggregation, the set of all states $\mathcal{S}$ is partitioned into $\mathcal{S}_1 \ldots \mathcal{S}_m$. A feature ϕ_j is then defined as:

$$\phi_j(s_i) = 1 \Leftrightarrow s_i \in \mathcal{S}_j.$$

A trivial example of an aggregation is when aggregates are singletons as $\mathcal{S}_i = \{s_i\}$. Such features allow to express an arbitrary heuristic function, but do not generalize the heuristic values.

Samples (1c) Samples are sequences of state-action pairs of one of the following types.

1. *Arbitrary* goal-terminated paths:

$$\Sigma_1 = (s_i^j, a_i^j)_{i=1}^{n_j} \quad s_{n_j} = \tau$$

   This set also contains all shorter sample paths that start with a state later in the sequence, which are also arbitrary goal-terminated samples.
2. *Optimal* goal-terminated paths:

$$\Sigma_2 = (s_i^j, a_i^j)_{i=1}^{n_j} \quad s_{n_j} = \tau$$

   We assume also that $\Sigma_2 \subseteq \Sigma_1$. Just as in Σ_1, all paths that start with a state later in the sequence are included in Σ_2.
3. *One-step* samples:

$$\tilde{\Sigma} = (s_1^j, a_1^j, s_2^j)$$

   We assume that $\tilde{\Sigma}$ also contain all transitions pairs from Σ_1 and Σ_2.

In general, the effort to needed gather an individual sample decreases from Σ_1 to $\tilde{\Sigma}$. However, the utility of the samples also decreases in the same way.

Samples may come either from previously solved problem instances or from randomly selected state transitions. Samples that come from previously solved problems are in the form Σ_1 or Σ_2. Randomly gathered samples are usually in the form $\tilde{\Sigma}$ and have a relatively low utility, but are much simpler to obtain. The availability of the various samples depends on the application. For example, it is rare to have optimal goal-terminated paths in online learning algorithms such as RTDP. Sampling is also more complicated in stochastic domains, such as Markov decision processes, because many samples may be required to determine the transition probabilities.

The inputs of the learning procedure determine its applicability and theoretical properties. Therefore, the procedures used in component (2) are classified according to their inputs as follows:

Inductive methods: Rely on state features (1b) or transition samples (1b) or both, but do not require a domain description. The methods usually cannot use the description even if it is provided. They are further classified as follows:

Feature-based methods: Only require features (1b) without using any samples. The methods are typically very simple and require high-quality features in order to produce admissible heuristic functions. The methods are described in detail in Section 11.3.1.

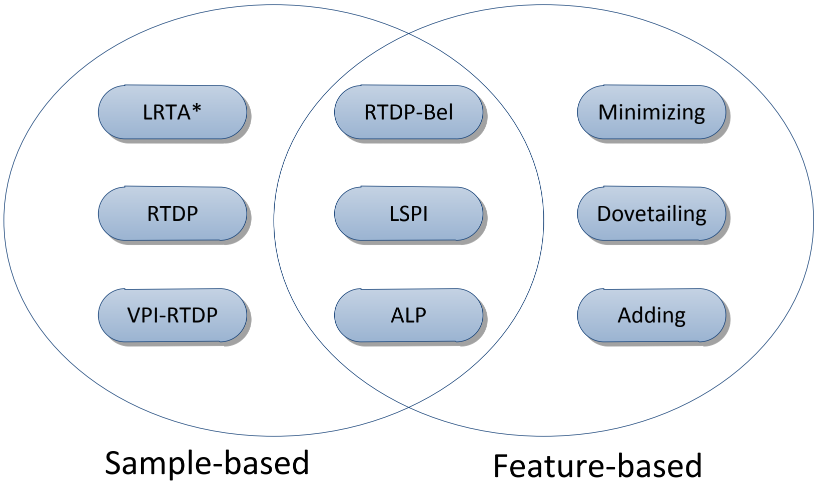

Figure 11.2. Examples of inductive methods based on the inputs they use.

- **Sample-based methods:** Use samples (1c) to learn the heuristic function for states that have been directly sampled. The methods cannot take advantage of features to generalize to unseen states. The methods are described in detail in Section 11.3.2.
- **Sample and feature-based methods:** Use samples (1c) to combine provided features (1b) to construct a heuristic value for states that have not been sampled. The features may be arbitrary, but the quality of the heuristic function highly depends on their quality. The methods are described in detail in Section 11.3.3.

Analytic methods: Rely on a domain description (1a), such as SAS+. These methods often cannot take advantage of features and samples, even if they are provided. They can be further classified as follows:

- **Abstraction-based methods:** Create a heuristic function without using samples or features. Instead, they simplify the problem using abstraction and solve it to obtain the heuristic function. The methods are described in detail in Section 11.3.4.
- **Other methods:** There is a broad range of analytic methods for many different problem formulations. A notable example is the simplification of mathematical programs by relaxations. For example, a linear program relaxation of an integer linear program defines an admissible heuristic function (Vanderbei, 2001). These relaxation methods have been also previously used in planning with success (Benton, van den Briel, & Kambhampati, 2007), but are not discussed in detail here.

The classification roughly follows Zimmerman and Kambhampati (2003). As mentioned above, the main focus of the chapter is on inductive methods. However, the inductive methods require features which may be hard to come by. The analytic methods, particularly those that are abstraction-based, can be a good source of the features.

Selected inductive heuristic learning methods are classified in Figure 11.2. The feature and sample-based method, lying at the intersection of the two sets, are most complex and interesting to study. The intersection contains also many reinforcement-learning methods, which are inherently incompatible with A*, as discussed in Section 11.3.3. Note, that the classification in the figure is with respect to the implementation of component (2) and it ignores the differences in other components.

11.3.1 Feature-based Methods

Feature based methods are inherently simple because the heuristic function is computed statically regardless of the specific problem structure. As a result, they are useful only when the features ϕ_i represent admissible heuristic functions. The three major feature-based methods are:

1. Minimum of heuristic functions[1]
2. Dovetailing
3. Sum of heuristic functions

Feature-based methods must use a *fixed* combination function $\bar{\theta}$ in (11.1) since there are no other inputs available.

The most common feature-based method is to simply take the minimum value of the features. The combination function $\bar{\theta}$ is defined as:

$$v(s) = \bar{\theta}(\phi_1(s), \ldots, \phi_k(s)) = \min\{\phi_1(s), \ldots, \phi_k(s)\},$$

The main advantage of this method is that if the features ϕ_i are admissible heuristic functions then the resulting heuristic function v is also admissible. *Dovetailing* is a refinement of the minimization which can reduce memory requirements by defining each feature only for subset of the state space (Culberson & Schaeffer, 1994).

The minimization method is simple but usually provides only a small improvement over the provided heuristic functions. An alternative is to statically add the available features:

$$v'(s) = \bar{\theta}(\phi_1(s), \ldots, \phi_k(s)) = \sum_{i=1}^{k} \phi_i(s).$$

There are, however, additional requirements in this case to ensure that v' is admissible. It is not sufficient to use arbitrary admissible heuristic functions as features. Designing suitable features is often complicated as has been shown in the area of *additive pattern databases* (Yang, Coulberson, Holte, Zahavi, & Felner, 2008).

11.3.2 Sample-based Methods

Sample-based methods are sometimes regarded as heuristic-search method, not as methods for learning heuristic functions. Nevertheless, they fit very well into the framework in Figure 11.1. They use samples to learn a heuristic function for the states visited online, but can be easily applied in offline settings as well. The main research question and the

[1] This would be a maximum in a cost minimization problem.

Algorithm 11.2: Real-Time Dynamic Programming (RTDP) see e.g. (Sanner, Goetschalckx, Driessens, & Shani, 2009)

```
Input: $v$ — an initial heuristic
Input: $\Sigma_1$ — arbitrary goal terminated samples
$v(s) \leftarrow \hat{v}(s)\ \forall s$ ;                                  /* Component (1c) */
while Convergence criterion is not met do
    visited.Clear() ;
    Pick initial state $s$ ;
    while $s \neq \tau$ do
        visited.Push($s$);
        $v(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma v(s)$ ;                /* Component (2) */
        $s \leftarrow$ ChooseNextState($s$, $h$) ;        /* Component (3), greedy in RTDP */
    /* Component (2):                                                         */
    while ¬ visited.IsEmpty do
        $s \leftarrow$ visited.Pop() ;
        $v(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma v(s)$ ;
```

differences between the algorithms relate to the selection of the samples, which is usually guided by some predetermined heuristic function.

The two main sample-based methods are LRTA* (Korf, 1988) and RTDP (Barto et al., 1995). RTDP can be seen as a generalization of LRTA* and therefore we focus on it and its many refinements. The section also includes a comparison with a significant body of theoretical work on sampling algorithms in reinforcement learning such as E^3 or R-Max.

RTDP, shown in algorithm 11.2, refines the heuristic values of the individual states by a backward propagation of heuristic values, essentially using value iteration (Puterman, 2005). The method builds a table of visited states with values $v(s)$. When a state s is expanded with children $s_1 \ldots s_k$, it is stored in a table along with the value

$$v(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma v(s).$$

When the function $v(s)$ is initialized to an upper bound on v^* RTDP will converge in the limit to $v(s) = v^*(s)$ for all visited states.

Modifications of RTDP may significantly improve its performance. Some notable RTDP extensions are Labeled RTDP (Bonet & Geffner, 2003b), HDP (Bonet & Geffner, 2003a), Bounded RTDP (McMahan, Likhachev, & Gordon, 2005), Focused RTDP (Smith & Simmons, 2006), and VPI-RTDP (Sanner et al., 2009). The algorithms differ in how the next states are chosen in "ChooseNextState", and in the convergence criterion. The procedure "ChooseNextState" is modified from greedy to one that promotes more exploration in the right places. Bounded RTDP, Focused RTDP, and VPI-RTDP determine the exploration by additionally using a lower bound on the value v^*. Note that v represents an upper bound on v^*.

The LAO* algorithm, described also in Section 11.2, can be seen as a sample-based method for learning heuristic functions. LAO* is based on policy iteration (Puterman, 2005), unlike value-iteration-based RTDP.

Many similar methods exist in reinforcement learning, such as R-Max (Brafman & Tennenholtz, 2002), E^3 (Kearns & Singh, 2002), UCT (Kocsis & Szepesvri, 2006), UCRL2 (Auer et al., 2009). Just like RTDP, they modify sample-selection in value iteration, but their motivation is quite different. In particular, the RL methods differ in the following ways: 1) they assume that the model is unknown, and 2) they have a well-defined measure of optimality, namely *regret*. Overall, RTDP focuses on the *optimization* part while the RL methods focus on the *learning* part. That is, while the RL methods aim to work almost as well as value iteration in *unknown* domains, RTDP methods attempt to improve on value iteration in *fully known* domains.

The difficulty with the purely sample-based methods is that the heuristic function is only calculated for sampled states. Generalization in RTDP could be achieved by using state features to represent the function $v(s)$. Regular RTDP can be seen as using features $\phi_1 \ldots \phi_n$ defined as:

$$\phi_i(s_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases},$$

which are combined linearly. It would be possible to extend RTDP to use arbitrary features, which would effectively turn it into Approximate Value Iteration (AVI). AVI is a common method in reinforcement learning but is in general unsuitable for constructing heuristic functions to be used with A*, as we discuss in Section 11.3.3.

11.3.3 Feature and Sample-based Methods

Feature and sample-based methods use the features to generalize from the visited state and the samples to combine the features adaptively. These methods are most general and most complex among the inductive heuristic learning methods.

As mentioned above, the heuristic function v is constructed using a combination function θ from some predetermined set Θ:

$$v(s) = \bar{\theta}(\phi_1(s), \phi_2(s), \ldots, \phi_k(s)).$$

The objective is to determine the $\bar{\theta} \in \Theta$ that would result in the best heuristic function. That is, for some measure of heuristic quality $\omega : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}$, the heuristic selection problem is:

$$\bar{\theta} = \operatorname*{argmin}_{\theta \in \Theta} \omega\left(\theta(\phi_1(\cdot), \phi_2(\cdot), \ldots, \phi_k(\cdot))\right).$$

The objective function ω is chosen as discussed in Section 11.2.2.

The space of combination functions Θ is typically restricted to linear functions for simplicity. The heuristic function is then expressed as:

$$v(s) = \theta(\phi_1(s), \phi_2(s), \ldots, \phi_k(s)) = \sum_{i=1}^{k} x_i \phi_i(s),$$

and the construction of the heuristic function becomes:

$$\bar{\theta} = \operatorname*{arg\,min}_{x \in \mathbb{R}^k} \omega\left(\sum_{i=1}^{k} x_i \phi_i(\cdot)\right).$$

That is, the objective is to determine the weights x_i for all features. As described in detail below, these methods are closely related to linear value function approximation in reinforcement learning (Powell, 2007a; Sutton & Barto, 1998). The linear feature combination in the vector form is:

$$v = \Phi x,$$

where Φ is a matrix that represents the features as explained in Section 11.4. The ordering of the states and actions is arbitrary but fixed.

The samples in feature and sample-based methods may be incomplete. That is, there are potentially many transitions that are not sampled. The samples are also incomplete in classical planning when the search problem is fully known. The limited set of samples simplifies solving the problem. This is in contrast with reinforcement learning in which more samples cannot be obtained. The presence of the model opens up the possibility of choosing the "right" samples.

Few feature and sample-based methods have been proposed within classical planning. The symbolic RTDP method (Feng et al., 2003) enables RTDP to generalize beyond the visited states in some domains. Yoon et al. (2008) propose a method that calculates a heuristic function from a linear combination of state features. The coefficients x are calculated using an iterative regression-like method. They also propose to construct state features from relaxed plans, obtained by deleting some of the preconditions in actions. Another feature and sample-based method, RTDP-Bel, was proposed in the context of solving POMDPs (Bonet & Geffner, 2009). RTDP-Bel endows RTDP with a specific set of features, which allow generalization in the infinite belief space of a POMDP. The resulting heuristic function is not guaranteed to be admissible.

Reinforcement learning The notion of a heuristic function in search is very similar to the notion of value function in reinforcement learning (Sutton & Barto, 1998). Value function is also maps states to real values that represent the expected sum of discounted reward achievable from that state. In fact, the value function in an approximation of v^*. The objective of most methods in reinforcement learning is to learn the value function as a combination of provided features and a set of transition samples using approximate dynamic programming (Powell, 2007a). The policy is then calculated from the value function using greedy search.

Approximate Dynamic Programming (ADP) fits well into the category of feature and sample-based methods. It has also been used in some interesting planning settings (Dzeroski, de Raedt, & Driessens, 2001; Fern, Yoon, & Givan, 2006). Yet, there is a significant incompatibility between ADP and classical planning applications. The policy in ADP is calculated using greedy best-first search, while in classical learning it is calculated using heuristic search. This results in two main difficulties with using ADP for learning heuristic functions: (1) ADP calculates functions that minimize the Bellman residual, which has little influence on the quality of the heuristic, and (2) they do not consider admissibility.

As Section 11.2.2 shows, there are difficulties with assessing the quality of a heuristic function when used with A*. It is much simpler if the heuristic function is instead used with greedy search. Greedy search has a fixed time complexity and the heuristic influences only the quality of the policy. The quality of the policy is bounded by the Bellman residual of v. The Bellman residual $\mathcal{B}(s)$ for a state s and a heuristic function v is defined as:

$$\mathcal{B}(s) = v(s) - \max_{a \in \mathcal{A}} \left(v(P(s,a)) - r(s,a) \right).$$

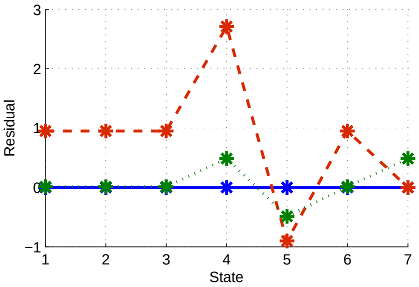

Figure 11.3. Bellman residual of three heuristic functions for a simple chain problem

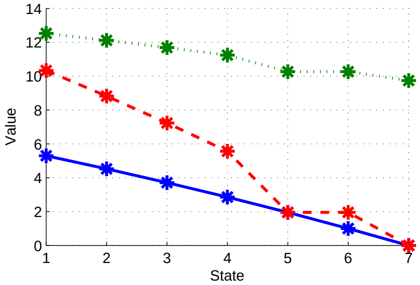

Figure 11.4. Three value functions for a simple chain problem

The quality of the greedy solution is a function of (Munos, 2003):

$$\frac{\max_{s \in \mathcal{S}} \mathcal{B}(s)}{1-\gamma}.$$

The bounds requires a discount factor $\gamma < 1$, which is not the case on most planning problems.

Most approximate dynamic programming methods minimize a function of the Bellman residual to get a good greedy policy. A heuristic function with a small Bellman residual may be used with A*, but it may be very far away from the true heuristic function. In particular, adding a constant to the heuristic changes the Bellman residual by factor of $1-\gamma$, which is very small when $\gamma \to 1$, but significantly increases the number of nodes expanded by A*. Figures 11.3 and 11.4 illustrate this issue on a simple test problem. The horizontal axes represent the state space and vertical axes represent respectively the Bellman residual and value of the corresponding state. The solid line is the optimal value function v^*. Notice that the dotted function has a very small Bellman residual, but a large true error. On the other hand, the dashed function has a large Bellman residual, but the error of the value is small.

Because of the above-mentioned issues, most reinforcement learning methods are unsuitable for calculating heuristic functions to be used with A*. One notable exception is approximate linear programming which is described in detail in Section 11.4. Approximate linear programming does not minimize the Bellman residual but bound on the error of the heuristic function.

The natural question in feature-based methods is the source of the features. Unfortunately, there is no good answer yet and the methods are usually domain-specific. Often, existing heuristic functions for the domain are a good source of features. This is practical when obtaining heuristic functions in a domain is easy, such as when analytic methods are applicable.

11.3.4 Abstraction-based Methods

Abstraction-based methods can quickly generate many relevant heuristic functions in domains with a suitable structure. They work by solving a simplified version of the problem, and can be traced at least to work by Samuel (Samuel, 1959). There has been significant progress in recent years automatic construction of heuristic functions using state abstraction in either deterministic (Holte, Mkadmi, Zimmer, & MacDonald, 1996a) or stochastic domains (Beliaeva & Zilberstein, 2005). Effective methods have been developed based on hierarchical heuristic search (Holte, Grajkowski, & Tanner, 2005) and pattern databases (Culberson & Schaeffer, 1996). These method are often able to solve challenging problems that were unsolvable before.

Abstraction-based approaches can be classified into two main types as follows (Valtorta, 1984):

Homomorphism Additional transitions among states are allowed. This leads to a shorter path to the goal and an admissible heuristic function.

Embedding Sets of states are grouped together and treated as a single state. This leads to a much smaller problem, which can be solved optimally. The solution of the simplified problem represents an admissible heuristic function.

A key advantage of abstraction-based methods is that they guarantee admissibility of the heuristic function.

Formally, when using abstraction on a search problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, P, r)$, a new search problem $\mathcal{P}' = (\mathcal{S}', \mathcal{A}', P', r')$ is defined, with a mapping function $b : \mathcal{S} - > \mathcal{S}'$. The optimal value functions v^* and $v^{*\prime}$ must satisfy for all $s \in \mathcal{S}$:

$$v^{*\prime}(b(s)) \geq v^*(s).$$

That means that the optimal value function in $\mathcal{P}'$ represents an admissible heuristic function in $\mathcal{P}$.

Abstraction methods can also be defined formally as follows. An abstraction is called a *homomorphism* when:

$$\begin{aligned} \mathcal{S} &= \mathcal{S}' & & \\ b(s) &= s & & \forall s \in \mathcal{S} \\ P(s,a) = s' &\Rightarrow P'(s,a) = s' & & \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned}$$

An abstraction is called an *embedding* when:

$$\begin{aligned} |\mathcal{S}'| &\leq |\mathcal{S}| & & \\ P(s,a) = s' &\Leftrightarrow P'(b(s),a) = b(s') & & \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned}$$

Although abstraction can be defined for arbitrary search spaces, additional structure is required to make it efficient. Such structure is can be provided by a description using a logic-based language.

Embedding provides two main advantages over homomorphism. First, assume that a search problem $\mathcal{P}$ is abstracted using *homomorphism* into $\mathcal{P}'$ and solved using blind search to obtain a heuristic v'. Then, the combined effort of blindly solving $\mathcal{P}'$ and solving $\mathcal{P}$ using A* with v' equals to solving $\mathcal{P}$ using *blind search* (Valtorta, 1984). In such settings, homomorphism provably does not provide any improvement. Second, a heuristic function computed using homomorphism may be difficult to store, while embedding provides a natural method for storing the heuristic function. Because the number of abstract states is typically small, the heuristic function is easy to store and to reuse over multiple searches. As a result, homomorphism is useful only when the relaxed value may be calculated faster than using blind search.

Despite difficulties with homomorphism, it has been used successfully in general purpose planning by Yoon et al. (2008). While they calculate the heuristic function is using a form of blind search, the policy is calculated from the heuristic function using a greedy search. As a result, Valtorta theorem does not apply. The method leads to good results since the constructed heuristic function is sufficiently accurate.

Embedding is very useful in general-purpose planning, because it is easily applied to domains represented using logic relations (Holte, Perez, R.M.Zimmer, & MacDonald, 1996b; Holte et al., 2005). Such languages are discussed in Section 11.3. Embeddings in problems described by logic languages are created by simply ignoring selected predicates (Edelkamp, 2002, 2007). Embedding used in structured problems is often known as *pattern databases* (Culberson & Schaeffer, 1996, 1998; Edelkamp, 2001).

When designing a pattern database, it is crucial to properly choose the right abstraction. This is a challenging problem, but some recent progress has been made recently (Drager, Fingbeiner, & Podelski, 2006; Edelkamp, 2006; Haslum, Bonet, & Geffner, 2005; Haslum, Botea, Helmert, Bonet, & Koenig, 2007; Helmert & Mattmuller, 2008). The existing methods use a local search in the space of potential abstraction to determine the best one. They often have good empirical results in specific domains, but their behavior is not well-understood and is very hard to analyze.

Because it is easy to obtain many pattern databases from different abstractions, it is desirable to be able combine them into a single heuristic function. Research on *additive pattern databases* tried to develop pattern databases that can be combined using the simple additive feature-based method (Yang et al., 2008), described in Section 11.3.1. Additive pattern databases guarantee that the sum of the heuristic function produces an admissible heuristic function, but constructing them is non-trivial. Therefore, combining pattern databases using feature and sample-based methods seems to be more promising.

11.4 Feature Combination as a Linear Program

The section describes in detail a method for learning heuristic functions based on approximate linear programming (ALP) (Petrik & Zilberstein, 2008). ALP, a common method in reinforcement learning, uses a linear program to calculate the heuristic function. The ALP approach can be classified as feature and sample-based and is related to methods described in Section 11.3.3. However, unlike most other feature and sample-based methods, ALP is guaranteed to produce admissible heuristic functions.

Linear programming, the basis for the ALP formulation, is a mathematical optimization method in which the solution is constrained by a set of linear inequalities. The optimal solution then minimizes a linear function of the set of constraints. A general form of a linear program is:

$$\begin{aligned} \min_{x} \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \geq b \end{aligned} \tag{11.2}$$

where c is the objective function and $Ax \geq b$ are the constraints. Linear programs can express a large variety of optimization problems and can be solved efficiently (Vanderbei, 2001).

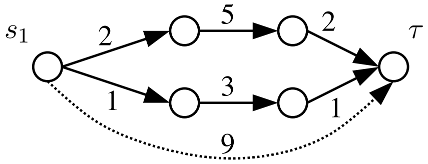

Figure 11.5. Formulations ensuring admissibility.

The heuristic function in ALP is obtained as a *linear* combination of the features ϕ_i. The set of feasible heuristic functions then forms a linear vector space $\mathcal{M}$, spanned by the columns of matrix Φ, which is defined as follows:

$$\Phi = \begin{pmatrix} \phi_1(s_1) & \phi_2(s_1) & \cdots \\ \phi_1(s_2) & \phi_2(s_2) & \cdots \\ & \vdots & \end{pmatrix}.$$

That is, each row of Φ defines the features for the corresponding state. The heuristic function may then be expressed as:

$$v = \Phi x,$$

for some vector x. Vector x represents the weights on the features, which are computed by solving the ALP problem. An important issue in the analysis is the representability of functions using the features, defined as follows.

Definition 11.2. A heuristic function, v, is *representable* in $\mathcal{M}$ if $v \in \mathcal{M} \subseteq \mathbb{R}^{|\mathcal{S}|}$, i.e. there exists such a z that $v = \Phi z$.

It is important to use a relatively small number of features in an ALP because of two main reasons. First, it makes the linear programs easy to solve. Second, it allows to use a small sample of all ALP constraint. This is important, since the total number of ALP constraints is greater than the number of all states in the search problem. Constraint sampling is explained in detail later in the section.

The remainder of the section describes multiple ALP formulations and their trade-offs. Section 11.4.1 describes two formulations of the constraints that ensure the admissibility of v. Section 11.4.2 introduces a new function θ that represents a *lower* bound on v^*. The difference between v and θ can be used to determine the accuracy of the heuristic function v. The difference is then used when formulating the linear program, as described in Section 11.4.3.

11.4.1 Upper Bound Constraints

This section describes two sets of linear constraints that can ensure the admissibility of the heuristic function. The feasible set is represented by a set of linear inequalities.

The two basic formulations are depicted in Figure 11.5. The first formulation, is to simply bound the heuristic value by the value received in the sampled states, and is represented by the dotted line in the figure. Formally, this is stated as:

$$h(s_i^j) \geq \sum_{k=i}^{n_j} \gamma^{k-i} r(s_k^j, a_k^j) \quad \forall j \in \Sigma_2,\, \forall i \in 1 \ldots n_j \tag{11.3}$$

Clearly, this formulation ensures the admissibility of the heuristic function. Notice that this will possibly mean multiple inequalities for each state, but only the dominating ones need to be retained. Thus let v_i denote the largest right-hand side for state s_i. The function must be restricted to the vector subspace spanned by columns of Φ. For notational convenience, we formulate the problem in a way that is independent of samples. Let v and v^* be column vectors with each row corresponding to a state. Then the inequality may be written as:

$$v = \Phi x \geq v^*,$$

treating v as a vector. In general, only some of the inequalities are provided based on the available samples; with all samples $v = v^*$. To simplify the notation, we denote this feasible set as $\mathcal{K}_1$, and thus $v \in \mathcal{K}_1$.

The second formulation is based on approximate linear programming, and is represented by the solid lines in Figure 11.5. In this case, the sample paths do not need to be terminated by the goal node. However, the heuristic function is actually required to be *consistent*, which is a stronger condition than admissibility. That is, for each observed sequence of two states, the difference between their heuristic values must be greater than the reward received. Formally,

$$\begin{aligned} &h(s_i^j) \geq \gamma v(s_{i+1}^j) + r(s_i^j, a_i^j) \quad \forall j \in \Sigma_2,\, \forall i \in 1 \ldots (n_j - 1) \\ &h(\tau) \geq 0 \end{aligned} \tag{11.4}$$

In this case, we can define an action-transition matrix T_a for action a. The matrix captures whether it is possible to move from the state defined by the row to the state defined by the column.

$$T_a(i,j) = 1 \Leftrightarrow t(s_i, a) = s_j.$$

A transition matrix T for all actions can then be created by vertically appending these matrices as follows:

$$T = \begin{pmatrix} T_{a_1} \\ \vdots \end{pmatrix}.$$

Similarly, we define a vector r_a of all the rewards for action a, such that $r_a(i) = r(s_i, a)$. The vector r of all the rewards for all the actions can then be created by appending the vectors:

$$r = \begin{pmatrix} r_{a_1} \\ \vdots \end{pmatrix}.$$

The constraints on the heuristic function in matrix form become:

$$v \geq \gamma T_a v + r_a \quad \forall a \in \mathcal{A},$$

together with the constraint $v(\tau) \geq 0$. To include the basis Φ to which the heuristic function is constrained, the problem is formulated as:

$$\begin{aligned}(\mathbf{I} - \gamma T_a)\Phi x &\geq r_a \quad \forall a \in \mathcal{A} \\ h(\tau) &\geq 0\end{aligned}$$

To simplify the notation, we denote the feasible set as $\mathcal{K}_2$, and thus $v \in \mathcal{K}_2$.

The formulation in (11.4) ensures that the resulting heuristic function will be admissible.

Proposition 11.3. *Given a complete set of samples, the heuristic function $v \in \mathcal{K}_2$ is admissible. That is, for all $s \in \mathcal{S}$, $v(s) \geq v^*(s)$.*

The proof of the proposition can be found in Section C.12.

In addition to admissibility, given incomplete samples, the heuristic function obtained from (11.4) is guaranteed not to be lower than the lowest heuristic value feasible in (11.3), as the following proposition states.

Proposition 11.4. *Let Σ_2 be a set of samples that does not necessarily cover all states. If v is infeasible in* (11.3)*, then v is also infeasible in* (11.4)*.*

The proof of this proposition is simple and relies on the fact that if an inequality is added for every segment of a path that connects it to the goal, then the value in this state cannot be less than the sum of the transition rewards.

Proposition 11.4 shows that given a fixed set of samples, (11.4) guarantees admissibility whenever (11.3) does. However, as we show below, it may also lead to a greater approximation error. We therefore analyze a hybrid formulation, weighted by a constant α:

$$\begin{aligned}&\forall j \in \Sigma_2, \forall i \in 1 \ldots (n_j - 1): \\ &\quad v(s_i^j) \geq \; \alpha\gamma v(s_{i+1}^j) + \alpha r(s_i^j, a_i^j) + (1-\alpha)v^*(s_i^j)\end{aligned} \tag{11.5}$$

Here $v^*(s_i^j)$ is the value of state s_i^j in sequence j. When it is not available, an arbitrary lower bound may be used. For $\alpha = 0$, this formulation is equivalent to (11.3), and for $\alpha = 1$, the formulation is equivalent to (11.4). We denote the feasible set as $\mathcal{K}_3$, and thus $v \in \mathcal{K}_3$. The key property of this formulation is stated in the following lemma, which is used later in the chapter to establish approximation bounds, and is straightforward to prove.

Lemma 11.5. *The optimal value function v^* is a feasible solution of* (11.5) *for an arbitrary α.*

11.4.2 Lower Bound Constraints

This section shows how to obtain a lower bound on the value of each state. This is important because it allows us to evaluate the difference between the heuristic value and the true value of each state. The lower bounds on the values of some selected states are obtained from the optimal solutions.

The formulation we consider is similar to (11.3).

$$\theta(s_i^j) \leq \sum_{k=i}^{n_j} \gamma^{k-i} r(s_k^j, a_k^j) \quad \forall j \in \Sigma_1, \forall i \in 1 \dots n_j \tag{11.6}$$

That is, the bounds are on the values of states that were solved optimally and any nodes that are on the path connecting the start state with the goal state. These bounds can also be written in matrix notation, as in (11.3):

$$\theta = \Phi y \geq v^*.$$

We denote this feasible set G_1, and thus $\theta \in G_1$. Additional bounds may be introduced as well. Given an admissible heuristic function, bounds can be deduced for any state that is expanded, even when it is not on an optimal path. While these bounds may not be tight in many cases, they will only increase the probability that the function θ is a lower bound. Notice that these constraints are sampled in the same manner as the constraints that ensure feasibility.

Proposition 11.6. *When the set of samples is complete and θ satisfies* (11.6), *then*

$$\theta(s) \leq v^*(s) \quad \forall s \in \mathcal{S}.$$

The proof of this proposition is straightforward.

In addition to the formulation above, a variation of (11.4) can also be considered. For this, assume that every state is reachable from the initial state σ. Then, the bounds can be written for $\forall j \in \Sigma_1$, $\forall i \in 1 \dots (n_j - 1)$ as:

$$\begin{aligned} \theta(s_{i+1}^j) \leq & \quad \gamma\theta(s_i^j) - r(s_i^j, a_i^j) \\ \theta(\sigma) \leq & v^*(\sigma). \end{aligned} \tag{11.7}$$

Unlike (11.4), these constraints alone do not guarantee that the function θ will be a lower bound on the optimal value of the states. Figure 11.6 depicts a situation in which these bounds are satisfied, but there is a feasible solution that is not an upper bound. Similarly, as in (11.4), the bounds may be formulated as:

$$\begin{gathered} (\mathbf{I} - \gamma T_a)\Phi y \geq r_a \quad \forall a \in \mathcal{A} \\ \theta(\sigma) \leq v^*(\sigma) \end{gathered}$$

We denote this feasible set G_2, and thus $\theta \in G_2$.

11.4.3 Linear Program Formulation

Given the above, we are ready to formulate the linear program for an admissible heuristic function. As discussed in Section 11.2.2, two simple metrics used for judging the quality of a heuristic function are the L_1 norm and L_∞ norm. Linear program formulations for each of the norm follow.

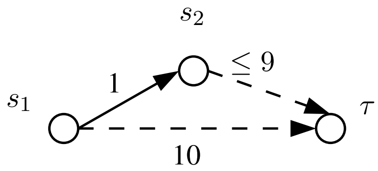

Figure 11.6. Lower bound formulations, where the dotted lines represent paths of arbitrary length.

The linear program that minimizes the L_1 norm of the heuristic function error is the following:

$$\begin{aligned} \min_{v} \quad & \mathbf{1}^{\mathsf{T}} v \\ \text{s.t.} \quad & v \in \mathcal{K}_3 \end{aligned} \tag{11.8}$$

The formulation corresponds exactly to approximate linear programming when $\alpha = 1$. It is easy to show that the optimal solution of (11.8) minimizes $\|v - v^*\|_1$ (de Farias, 2002). But to A linear program that minimizes the L_∞ norm of the heuristic function error is the following:

$$\begin{aligned} \min_{\delta, v} \quad & \delta \\ \text{s.t.} \quad & v(s) - \theta(s) \leq \delta \quad \forall s \in S \\ & v \in \mathcal{K}_3 \\ & \theta \in G_1 \end{aligned} \tag{11.9}$$

It is easy to show that (11.9) minimizes $\|v - v^*\|_\infty$. This is because from the definition $v(s) \geq \theta(s)$ for all s. In addition, even when the linear program is constructed from the samples only, this inequality holds. Notice that the number of constraints $v(s) - \theta(s) \leq \delta$ is too large, because one constraint is needed for each state. Therefore, in practice these constraints will be sampled as well as the remaining states. In particular, we use those states s for which $v^*(s)$ is known. While it is possible to use G_2 instead of G_1, that somewhat complicates the analysis. We summarize below the main reasons why the formulation in (11.9) is more suitable than (11.8).

11.5 Approximation Bounds

We showed above how to formulate the linear programs for optimizing the heuristic function. It is however important whether these linear programs are feasible and whether their solutions are close to the best heuristic that can be represented using the features in basis Φ. In this section, we extend the analysis used in approximate linear programming to show new conditions for obtaining a good heuristic function.

We are interested in bounding the maximal approximation error $\|v - v^*\|_\infty$. This bound limits the maximal error in any state, and can be used as a rough measure of the extra search effort required to find the optimal solution. Alternatively, given that $\|v - v^*\|_\infty \leq \epsilon$, then the greedily constructed solution with this heuristic will have the approximation error of at most $m\epsilon$, where m is the number of steps required to reach the goal. This makes it possible to solve the problem without search. For simplicity, we do not address here the issues related to limited sample availability, which have been previously analyzed (de Farias, 2002; de Farias & van Roy, 2004; Ben-Tal & Nemirovski, 2008; Goldfarb & Iyengar, 2003)

The approximation bound for the solution of (11.8) with the constraints in (11.4) comes from approximate linear programming (de Farias, 2002). In the following, we use **1** to denote the vector of all ones. Assuming **1** is representable in $\mathcal{M}$, the bound is:

$$\|v^* - v\|_c \leq \frac{2}{1-\gamma} \min_x \|v^* - \Phi x\|_\infty,$$

where $\|\cdot\|_c$ is an L_1 error bound weighted by a vector c, elements of which sum to 1. The approximation bound contains the multiplicative factors, because even when Φx is close to v^* it may not satisfy the required feasibility conditions. This bound only ensures that the sum of the errors is small, but errors in some of the states may still be very large. The bound can be directly translated to an L_∞ bound, assuming that $c = \mathbf{1}$, that is a vector of all ones. The bound is as follows:

$$\|v^* - v\|_\infty \leq |\mathcal{S}| \frac{2}{1-\gamma} \min_x \|v^* - \Phi x\|_\infty.$$

The potential problem with this formulation is that it may be very loose when: (1) the number of states is large, since it depends on the number of states $|\mathcal{S}|$; or (2) the discount factor γ is close to 1 or is 1.

We show below how to address these problems using the alternative formulation of (11.9) and taking advantage of additional structure of the approximation space.

Lemma 11.7. *Assume that* **1** *is representable in* $\mathcal{M}$*. Then there exists a heuristic function* $\hat{v}$ *that is feasible in* (11.5) *and satisfies:*

$$\|\hat{v} - v^*\|_\infty \leq \frac{2}{1-\gamma\alpha} \min_x \|v^* - \Phi x\|_\infty.$$

The proof of the lemma can be found in Section C.12.

Using similar analysis, the following lemma can be shown.

Lemma 11.8. *Assume* **1** *is representable in* $\mathcal{M}$*. Then there exists a lower bound* $\hat{\theta}$ *that is feasible in* (11.6)*, such that:*

$$\|\hat{\theta} - v^*\|_\infty \leq 2 \min_x \|v^* - \Phi x\|_\infty.$$

This lemma can be proved simply by subtracting $\epsilon\mathbf{1}$ from θ that is closest to v^*. The above lemmas lead to the following theorem with respect to the formulation in (11.9).

Lemma 11.9. *Assume that* **1** *is representable in* $\mathcal{M}$*, and let* $\hat{v}, \hat{\theta}, \delta$ *be an optimal solution of* (11.9)*. Then:*

$$\delta = \|\hat{v} - v^*\|_\infty \leq \left(2 + \frac{2}{1-\gamma\alpha}\right) \min_x \|v^* - \Phi x\|_\infty.$$

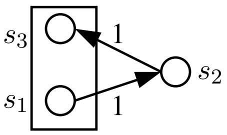

Figure 11.7. An approximation with loose bounds.

Therefore, by solving (11.9) instead of (11.8), the error is independent of the number of states. This is a significant difference, since the approach is proposed for problems with a very large number of states.

Proof. Assume that that the solution δ does not satisfy the inequality. Then, using Lemmas 11.8 and 11.7, it is possible to construct a solution $\hat{v}, \hat{\theta}, \hat{\delta}$. This leads to a contradiction, because $\hat{\delta} < \delta$. □ □

Even when (11.9) is solved, the approximation error depends on the factor $1/(1-\gamma\alpha)$. For $\gamma = \alpha = 1$, the bound is infinite. In fact the approximate linear program may become infeasible in this case, unless the approximation basis Φ satisfies some requirements. In the following, we show which requirements are necessary to ensure that there will always be a feasible solution.

To illustrate this problem with the approximation, consider the following simple example with states $\mathcal{S} = \{s_1, s_2, s_3\}$ and a single action $\mathcal{A} = \{a\}$. The goal is the state $\tau = s_3$, and thus there is no transition from this state. The transitions are $t(s_i, a) = s_{i+1}$, for $i = 1, 2$. The rewards are also $r(s_i, a) = 1$ for $i = 1, 2$. Now, let the approximation basis be

$$\Phi = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}^{\mathsf{T}}.$$

This example is depicted in Figure 11.7, in which the square represents the aggregated states in which the heuristic function is constant. The bounds of (11.4) in this example are

$$\begin{array}{rcl} h(s_1) & \geq & \gamma v(s_2) + 1 \\ h(s_2) & \geq & \gamma v(s_3) + 1 \\ h(s_3) & \geq & 0 \end{array}$$

The approximation basis Φ requires that $v(s_1) = v(s_3)$. Thus we get that:

$$v(s_1) \geq \gamma v(s_2) + 1 \geq \gamma^2 v(s_3) + \gamma + 1 = \gamma^2 v(s_1) + \gamma + 1.$$

As a result, despite the fact that $v^*(s_1) = 2$, the heuristic function is $v(s_2) = (1+\gamma)/(1-\gamma^2)$. This is very imprecise for high values of γ. A similar problem was addressed in standard approximate linear programming by introducing so called Lyapunov vectors. We build on this idea to define conditions that enable us to use (11.5) with high γ and α.

Definition 11.10 (Lyapunov vector hierarchy)**.** Let $u^1 \dots u^k \geq \mathbf{0}$ be a set of vectors, and T and r be partitioned into T_i and r_i respectively. This set of vectors is called a Lyapunov vector hierarchy if there exist $\beta_i < 1$ such that:

$$\begin{array}{rcl} T_i u^i & \leq & \beta_i u^i \\ T_j u^i & \leq & \mathbf{0} \quad \forall j < i \end{array}$$

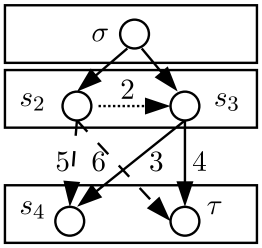

Figure 11.8. An example of the Lyapunov hierarchy. The dotted line represents a constraint that needs to be removed and replaced by the dashed ones.

The second condition requires that no states in partition j transit to a state with positive u^i.

An example of such a hierarchy would be an abstraction, depicted in Figure 11.8. Let the state space $\mathcal{S}$ be partitioned into l subsets $\mathcal{S}_i$, with $i \in 1 \ldots l$. Assume that the transitions satisfy:

$$\forall a \in \mathcal{A} : \quad t(s,a) = s' \wedge s \in \mathcal{S}_i \wedge s' \in \mathcal{S}_j \Rightarrow j < i.$$

That is, there is an ordering of the partitions consistent with the transitions. Let u^i be a vector of the size of the state space, defined as:

$$u^i(s) = 1 \Leftrightarrow s \in \mathcal{S}_i,$$

and zero otherwise. It is easy to show that these vectors satisfy the requirements of Definition 11.10. When the approximation basis Φ can be shown to contain such u^i, it is, as we show below, possible to use the formulation with $\gamma = \alpha = 1$ with low approximation error.

Lemma 11.11. *Assume that there exists a Lyapunov hierarchy $u^1 \ldots u^l$, such that each u^i is representable in $\mathcal{M}$. Then there exists a heuristic function $\hat{v}$ in Φ that is feasible in (11.5), such that:*

$$\|\hat{v} - v^*\|_\infty \le \left(\prod_{i=1}^{l} \frac{(1+\alpha\gamma)\max_{s\in\mathcal{S}} u^i(s)}{(1-\alpha\gamma\beta_i)\min_{s\in\mathcal{S}} u_i^i(s)} \right) 2 \min_x \|v^* - \Phi x\|_\infty,$$

where u_i^i is the vector u^i restricted to states in partition i.

The proof of the lemma can be found in Section C.12.

The bound on the approximation error of the optimal solution of (11.9) may be then restated as follows.

Theorem 11.12. *Assume that there exists a Lyapunov hierarchy $u^1 \ldots u^l$, and for each u^i there exists z_i such that $u^i = \Phi z_i$. Then for the optimal solution $\hat{v}, \delta$ of (11.9):*

$$\delta = \|\hat{v} - v^*\|_\infty \le \left(1 + \prod_{i=1}^{l} \frac{(1+\alpha\gamma)\max_{s\in\mathcal{S}} u^i(s)}{(1-\alpha\gamma\beta_i)\min_{s\in\mathcal{S}} u_i^i(s)} \right) 2\epsilon,$$

where $\epsilon = \min_x \|v^* - \Phi x\|_\infty$.

The proof follows from Lemma 11.11 similarly to Lemma 11.9. The theorem shows that even when $\gamma = 1$, it is possible to guarantee the feasibility of (11.9) by including the Lyapunov hierarchy in the basis.

A simple instance of a Lyapunov hierarchy is a set of features that depends on the number of steps from the goal. Therefore, the basis Φ must contain a vector u^i, such that $u^i(j) = 1$ if the number of steps to get to s_j is i and 0 otherwise. This is practical in problems in which the number of steps to the goal is known in any state. Assuming this simplified condition, Theorem 11.12, may be restated as follows.

$$\|\hat{v} - v^*\|_\infty \leq \left(1 + \prod_{i=1}^{l} \frac{1+\alpha\gamma}{1-\alpha\gamma\beta_i}\right) 2 \min_x \|v^* - \Phi x\|_\infty .$$

This however indicates an exponential growth in error with the size of the hierarchy with $\gamma = \alpha = 1$. Unfortunately, it is possible to construct an example in which this bound is tight. We have not observed such behavior in the experiments, and it is likely that finer error bounds could be established. As a result of the analysis above, if the basis contains the Lyapunov hierarchy, the approximation error is finite even for $\gamma = 1$.

In some problems it is hard to construct a basis that contains a Lyapunov hierarchy. An alternative approach is to include only constraints that obey the Lyapunov hierarchy present in the basis. These may include multi-step constraints, as indicated in Figure 11.8. As a result, only a subset of the constrains is added, but this may improve the approximation error significantly. Another option is to define features that depend on the actions, not only on states. This is a non-trivial extension, however, and we leave the details to future work. Finally, when all the rewards are negative and the basis contains only a Lyapunov hierarchy, then it can be shown that no constraints need to be removed.

11.6 Empirical Evaluation

We evaluate the approach on the sliding eight tile puzzle problem–a classic search problem (Reinefeld, 1993). The purpose of these experiments is to demonstrate the applicability of the ALP approach. We used the eight-puzzle particularly because it has been studied extensively and because it can be solved relatively quickly. This allows us to evaluate the quality of the heuristic functions we obtain in different settings. Since all the experiments we describe took less than a few seconds, scaling to large problems with many sample plans is very promising. Scalability mostly relies on the ability to solve efficiently large linear programs–an area that has seen significant progress over the years. In all instances, we use the formulation in (11.9) with different values of α.

The heuristic construction method relies on a set of features available for the domain. Good features are crucial for obtaining a useful heuristic function, since they must be able to discriminate states based on their heuristic value. In addition, the set of features must be limited to facilitate generalization. Notice that although the features are crucial, they are in general much easier to select compared with a good admissible heuristic function. We consider the following basis choices:

1. Manhattan distance of each tile from its goal position, including the empty tile. This results in 9 features that range in values from 0 to 4. This basis does not satisfy the Lyapunov property condition. The minimal admissible heuristic function from these

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	States
0	-1	-1	-1	-1	0	0	-1	0	958
0	-1	-1	-1	-1	-1	0	-1	0	70264
0	-1	-1	0	-1	-1	-1	-1	-1	63
0	-1	-1	-1	-1	-1	-1	-1	-1	162

Figure 11.9. Weights calculated for individual features using the first basis choice. Column x_i corresponds to the weight assigned to feature associated with tile i, where 0 is the empty tile. The top 2 rows are based on data from blind search, and the bottom 2 on data from search based on the heuristic from the previous row.

features will assign value -1 to the feature that corresponds to each tile, except the empty tile, which is 0.

2. Abstraction based on the sum of the Manhattan distances of all pieces. For example, feature 7 will be 1 if the sum of the Manhattan distances of the pieces is 7 and 0 otherwise. This basis satisfies the Lyapunov hierarchy condition as defined above, since all the rewards in the domain are negative.
3. The first feature is the Nilsson sequence score (Nilsson, 1971) and the second feature is the total Manhattan distance minus 3. The Nilsson sequence score is obtained by checking around the non-central square in turn, allotting 2 for every tile not followed by its proper successor and 0 for every other tile, except that a piece in the center scores 1. This value is not an admissible heuristic.

First, we evaluate the approach in terms of the number of samples that are required to learn a good heuristic function. We first collect samples from a blind search. To prevent expanding too many states, we start with initial states that are close to the goal. This is done by starting with the goal state and performing a sequence of 20 random actions. Typical results obtained in these experiments are shown in Figure 11.9, all performed with $\alpha = 1$. The column labeled "States" shows the total number of node-action pairs expanded and used to learn the heuristic function, not necessarily unique ones. The samples are gathered from solving the problem optimally for two states. The results show that relatively few nodes are required to obtain a heuristic function that is admissible, and very close to the optimal heuristic function with the given features. Notice that the heuristic function was obtained with no prior knowledge of the domain and without any a priori heuristic function. Very similar results were obtained with the second basis.

Next, we compare the two formulations for the upper bounds, (11.3) and (11.4), with regard to their approximation error. Notice that a big advantage of the formulation depicted by (11.4) is the ability to use transitions from states that are not on a path to the goal. The data is based on 100 goal terminated searches and 1000 additional randomly chosen states. The results are shown in Figure 11.10. Here, δ is the objective value of (11.9), which is the maximal overestimation of the heuristic function in the given samples. Similarly, δ' is the maximal overestimation obtained based on 1000 state samples independent of the linear program. The approximate fraction of states in which the heuristic function *is* admissible is denoted by p. These results demonstrate the tradeoff that α offers. A lower value of α generally leads to a better heuristic function, but at the expense of admissibility. The bounds with regard to the sampled constraints, as presented in (de Farias, 2002) do not distinguish between the two formulations. A deeper analysis of this is an important part of

α	1	0.99	0.9	0.8	0
x_1	0	-0.26	-1	-1	-1
x_2	-0.25	-0.25	-0.59	-0.6	-0.6
x_3	0	0	2.52	2.6	2.6
δ	17	16.49	13.65	13.6	13.6
p	1	1	0.97	0.96	0.95
δ'	19	17	15	14.4	14.4

Figure 11.10. The discovered heuristic functions as a function of α in the third basis choice, where x_i are the weights on the corresponding features in the order they are defined.

future work. Interestingly, the results show that the Nilsson sequence score is not admissible, but it becomes admissible when divided by 4.

The ALP approach has also been applied to Tetris, a popular benchmark problem in reinforcement learning (Szita & Lorincz, 2006). Because Tetris is an inherently stochastic problem, we used the Regret algorithm (Mercier & Hentenryck, 2007) to solve it using a deterministic method. The Regret algorithm first generates samples of the uncertain component of the problem and then treats it as a deterministic problem. It is crucial in Tetris to have a basis that contains a Lyapunov hierarchy, since the rewards are positive. Since we always solve the problem for a fixed number of steps forward, such a hierarchy can be defined based on the number of steps remaining as in Figure 11.8. Our initial experiments with Tetris produced promising results. However, to outperform the state-of-the-art methods–such as approximate linear programming (Farias & van Roy, 2006) and cross-entropy methods (Szita & Lorincz, 2006)–we need to scale up our implementation to handle millions of samples. This is mostly a technical challenge that can be addressed using methods developed by (Farias & van Roy, 2006).

11.7 Contributions

The main contribution described in this chapter is the connection between heuristic search and value function approximation. In particular, it shows how to use approximate linear programming to compute *admissible* heuristic functions. The chapter also presents a new classification and overview of methods for learning heuristic functions. These methods are similar to value function approximation but typically concern problems with more structure.

CHAPTER 12

CONCLUSION

This thesis proposed a new approach to value function approximation, or approximate dynamic programming, using optimization–based formulations. We examined various formulations that focus on optimizing diverse properties. Each of the formulations has its own advantages and disadvantages and may be, therefore, applicable in different circumstances. For example, the approximate linear programming formulation is relatively loose, but represents a convex optimization problem. Linear programs are easy to solve, for example using the proposed homotopy method, and are convenient to analyze. Approximate bilinear programs, on the other hand, guarantee much tighter approximation with very little extra prior knowledge. Unfortunately, bilinear programs are much harder to solve and analyze.

An important advantage of the optimization–based approach is that it *decouples* the formulation from the algorithms used to solve it. The formulation captures the properties of the value function that need to be attained. The formulations are in terms of generic mathematical optimization problems, for which many algorithms have been proposed. In addition, specific properties of the formulations can be used to develop more efficient solution methods. Nevertheless, it is possible to analyze the formulations without knowing the intricacies of the specific algorithm. It is also possible to develop solution algorithms without influencing the analysis of the formulation.

The practical benefits of the proposed methods in practice are yet to be fully evaluated. We applied the methods to a number of benchmark and practical problems, and got very encouraging results. However, the true practical evaluation of the optimization–based approach will require that the results are replicated in many other domains and settings.

The work on optimization–based methods for value function approximation is by no means complete. This thesis presents many basic results that will need to be significantly refined in order to become practical in wider array of settings. We mention a non–exhaustive list of significant topics that require future work:

Better bounds on policy loss Existing bounds on policy loss are often very loose. Since these bounds are at the heart of optimization–based methods, improving them could dramatically improve the performance of the methods. There has been very little work on policy loss bounds.

Policy visitation–frequencies approximation Some approximate linear and bilinear formulations rely heavily on being able to approximate the policy visitation frequencies. There are no known methods that can currently approximate the visitation frequencies. This is quite different than approximating value functions. The Bellman residual, which depends only on a single step in the MDP, can be used to evaluate the quality of the greedy policy with respect to the value function. No such concept exists for state visitation frequencies.

Approximate algorithms for solving approximate bilinear programs The approximate algorithms for solving bilinear programs presented in this thesis are quite limited. Further work is needed to explore potential approximate methods for solving bilinear programs with this structure. These algorithms can be based either on solving bilinear programs or on the mixed integer linear program formulation that we proposed.

Better sampling bounds The sampling bounds described in the thesis are quite loose in most circumstances. Additional problem–specific assumptions will be required in order to make the bounds useful.

Structure–based offline error bounds The bounds and the whole approach presented in the thesis ignores the structure of the problem and assumes that it can be captured by the features. While we present methods that work with very large and rich feature spaces, we did not address how the structure of the domain may influence the quality of the given features. That is, new offline bounds are needed — bounds that will not be with respect to the closest approximation of the optimal value function, but bounds that will be in terms of the MDP domain structure.

Optimization–based approximate dynamic programming presents a novel approach to approximately solving a large class of planning and learning problems. We believe that this approach opens up many opportunities to understand better the fundamental issues involved in value function approximation.

PART

APPENDICES

APPENDIX A

NOTATION

All vectors are assumed to be column vectors, unless specified otherwise. We use the following matrix and block matrix notation. Matrices are denoted by square brackets, with columns separated by commas and rows separated by semicolons. Columns have precedence over rows. For example, the notation $[A, B; C, D]$ corresponds to the matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$.

$\mathbf{1}$ Vector of all ones of the size appropriate to the context.

$\mathbf{1}_i$ Vector of all zeros except the i-th element, which is one, of the size appropriate to the context.

$\mathbf{0}$ Vector of all zeros of the size appropriate in the context.

α Initial-state distribution.

β Lyapunov-style constant used to derive bounds.

γ Discount factor in $[0, 1]$.

σ A bound on the violation of the transitive–feasible constraints when sampling.

π A policy in the given MDP.

Π A set of possible policies to a given MDP.

ρ The maximal difference between the upper bound on v^* and a lower bound, such as $\|h - v^*\|_\infty$ or $\|v - \theta\|_\infty$.

ρ The regression operator used for estimating the sampling error.

Σ Set of samples used to determine the approximate value function — generic samples.

$\bar{\Sigma}$ Set of informative samples that contain the transition distribution.

σ Initial state in search problems.

Σ_1, Σ_2 Samples used for computing heuristic functions.

$\phi(s)$ Features of state s. Represents a vector.

ϕ_i Feature i as a vector for all states.

Φ Approximation basis, represented as a matrix.

θ Lower bound on the optimal value function.

τ Goal state in search problems.

ξ Coefficients of the approximation features Φ for the lower bound on the value function, such that $\theta = \Phi\xi$.

χ Operator that maps states to samples and is used in estimating the sampling error.

θ_B Optimal objective value of the regularized ABP, as a function of ψ

ψ Regularization coefficient for regularized formulations: $\|x\|_{1,e} \leq \psi$.

k The state embedding function. Maps states to $\mathbb{R}^n$.

A Matrix that represents the constraints on transitive-feasible value functions.

$\mathcal{A}$ Set of available actions, uniform over all states.

B Matrix that represents the constraints on the stochasticity of policies.

c Objective function typically used in the approximate linear programming formulation. Alternatively, this is used also as an arbitrary distribution over the states in the MDP.

$\mathcal{C}$ Set of children of a state in a search problem.

d Distance (sum of rewards) between any two states in a search problem.

e Regularization weights for regularized formulations: $\|x\|_{1,e} \leq \psi$.

$\mathbf{I}$ Identity matrix of the size appropriate to the context.

$\mathcal{K}$ Set of transitive-feasible value functions.

L Bellman operator

G_i One of the value function feasibility sets that ensure that $v \leq v^*$.

n_j Length of sample j in terms of the number of states and actions.

$\mathcal{N}$ A (multivariate) normal distribution.

P_a A matrix that represents the probability of transiting from the state defined by the row to the state defined by the column, such that: $P_a(i,j) = P(s_i, a, s_j)$.

P Transition operator for an MDP.

P_π A matrix that represents the probability of transiting from the state defined by the row to the state defined by the column, such that: $P_\pi(i,j) = P(s_i, \pi(s_i), s_j)$.

r Vector of rewards in the MDP, defined for each state and action pair.

r_a Vector the rewards for action a, such that $r_a(i) = r(s_i, a)$.

r_π Vector the rewards for action a, such that $r_\pi(i) = r(s_i, \pi(s_i))$.

$\mathcal{S}$ Set of states of the problem.

P_π Transition function for a fixed policy.

u State-action visitation frequencies.

u_a Subset of state-action visitation frequencies for action a.

u_π State visitation frequencies for a policy π.

$\underline{\mathbf{u}}$,$\bar{u}$ Lower and upper bounds on u_π.

v Arbitrary value function, may represent a value function of a policy. We also use v to represent a heuristic function. While this notation is inconsistent with the search literature, it makes our treatment of the problem consistent.

$\tilde{v}$ An approximate value function.

v^* The optimal value function, such that $v^* \geq v$ for all v. This function is unique.

q State-action value function, also known as Q-function.

x Coefficients associated with the approximation features Φ, such that $v = \Phi x$. It is also used as a generic optimization variable for general linear programs.

$\mathcal{M}$ Set of representable value functions.

z The state transition function, used to define common random numbers.

APPENDIX B

PROBLEM DESCRIPTIONS

In this appendix, we briefly describe the problems that are used to evaluate the proposed formulations and algorithms. Most of these problems are relatively simple to facilitate easy computation of the optimal solutions and to minimize engineering challenges, which are in general independent of our work. We also, however, describe two problems that are based on real data and have a potential for deployment in the future.

B.1 Simple Benchmark Problems

B.1.1 One-directional Chain

This is a simple linear chain problem with 200 states, in which the transitions move to the right by one step with a centered Gaussian noise with standard deviation 3. The reward for reaching the right-most state was +1 and the reward in the 20th state was -3. This problem is small to enable calculation of the optimal value function and to control sampling.

The approximation basis in this problem is represented by piecewise linear features, of the form $\phi(s_i) = [i - c]_+$, for c from 1 to 200. The discount factor in the experiments was $\gamma = 0.95$.

B.1.2 Two-directional Chain

This is a simple linear chain problem with 200 states, in which the transitions move to the right or left (2 actions) by one step with a centered Gaussian noise of standard deviation 3. The rewards were set to $\sin(i/20)$ for the right action and $\cos(i/20)$ for the left action, where i is the index of the state. This problem is small enough to calculate the optimal value function and to control the approximation features.

The approximation basis in this problem is represented by piecewise linear features, of the form $\phi(s_i) = [i - c]_+$, for c from 1 to 200. The discount factor in the experiments was $\gamma = 0.95$ and the initial distribution was $\alpha(130) = 1$.

B.1.3 Mountain Car

In the mountain-car benchmark an underpowered car needs to climb a hill (Sutton & Barto, 1998). We use a modified version of this task. To do so, it first needs to back up to an

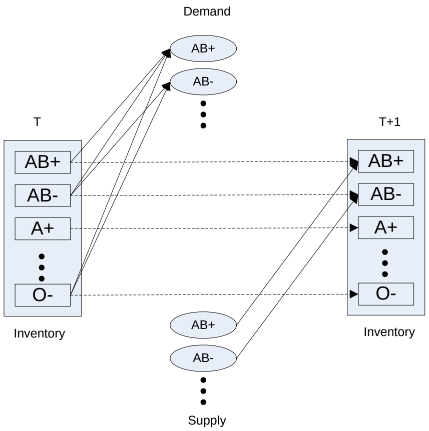

Figure B.1. A sketch of the blood inventory management

opposite hill to gain sufficient momentum. The state-space is $\mathcal{S} = \mathbb{R}^2$ and represents the position x_t and velocity y_t of the car. The dynamics of the environment are:

$$x_{t+1} = b_x(x_t + y_{t+1})$$
$$y_{t+1} = b_y(y_t + 0.001a_t + -0.0025cos(3x_t)).$$

Here, $b_x(x) = \min\{\max\{x, -1.2\}, 0.5\}$ and $b_y(y) = \min\{\max\{y, -0.07\}, 0.07\}$. The value a_t represents the action. The actions are $\mathcal{A} = \{-1, 0, 1\}$.

In the original formulation, the task stops as soon as the reward is received. In our modified problem, the car continues for an extra 0.1 distance. The dynamics of the problem do not allow it to receive the reward again, however. Formally, the reward is defined as:

$$r((x, y), a) = \begin{cases} 1 & \text{if } x > 0.4 \\ 0 & \text{otherwise} \end{cases}.$$

In this domain, a car is faced with a task of climbing a hill that is too steep for its engine. Therefore, to ascend the hill, it must first back up to an opposite hill to gain momentum. The problem is described by the car's position and velocity. The transitions are deterministic and guided by simplified laws of physics.

B.2 Blood Inventory Management

The blood inventory management problem concerns managing a blood supply inventory and determining the optimal blood-type substitution. Here, we provide only brief description of the basic model. For more detailed description and motivation, please see (Powell, 2007a). Note, however, that the formulation that we use is somewhat different to make the problem somewhat more challenging, as we detail below. The blood inventory management problem is sketched in Figure B.1.

Blood inventory management is a multi-dimensional resource management problem. The blood allocation decision is made weekly. The state-space represents the inventory of blood,

Type	Critical	Urgent	Elective
Unsatisfied	0	0	0
Same type	50	25	5
Compatible type	45	27.5	4.5

Figure B.2. Rewards for satisfying blood demand.

divided into 8 blood types (A+,A-,B+,B-,AB+,AB-,0+,0-) and 4 ages, and the blood demands for each blood type. A state-space is therefore $\mathcal{S} = \mathbb{R}^40$. The actions represent the assignment of blood to the demand and is: $\mathcal{A} = \mathbb{R}^{(40+8)}$. That means for each blood type in the inventory the amount used and for each demand the amount satisfied. The transition from state to state adds stochastic supply and generates new stochastic demand. For more details on the formulation and blood type compatibilities, please see (Powell, 2007a).

The problem that we consider is a slightly modified version of the originally proposed model. The myopic solution of the original model is about 273000 and easy to obtain upper bound on the optimal solution is 275000. This indicates that the possible improvement from advanced planning is relatively small. The main reason is that the blood supply is almost entirely capable of satisfying the demand.

To make the problem more interesting as a benchmark domain, we restrict the supply by 50% and add priorities for blood supply. We also introduce priorities for blood demands: 1) critical, 2) urgent, and 3) elective. The rewards for satisfying the demand, based on the priority and blood compatibility are summarized in Figure B.2. For this formulation, the myopic solution is about 70000 while the bound on the optimal one is about 94000.

This problem presents an interesting difficulty when compared to most other reinforcement learning applications. It has an infinite state-space and therefore simply computing the greedy policy for a value function is difficult. To make the computation possible, the we need to approximate the state–action value function q (such as in (4.3)) and it must be approximated using piecewise linear functions, which are described in Section 10.2.

The myopic solution represents the greedy policy for $v = 0$. We describe this solution first for simplicity. The action for every states is computed using the following linear program:

$$\begin{aligned} \max_{y,z} \quad & \sum_{ij} r_{ij} y_{ij} \\ \text{s.t.} \quad & \sum_{j} y_{ij} + z_k \leq C(i) \quad \forall i \in \mathcal{T} \\ & \sum_{i} y_{ij} \leq D(j) \quad \forall j \in \mathcal{T} \\ & y_{ij}, z_i \geq 0 \quad \forall i, j \in \mathcal{T} \end{aligned}$$

This maximum-flow linear program is sketched in Figure B.3. Here, indices i represent the blood types in the inventory and indices j represent the blood types in the demand. The set T denotes all blood types (for the sake of simplicity we ignore the age of the blood in the inventory). Then, $r(i, j)$ is the reward for using blood type (and age) i to satisfy demand j. $D(j)$ is the demand for blood type j, $C(j)$ is the amount of blood in the inventory, y_{ij} is the amount of blood used to satisfy the demand and z_k is the blood amount carried over to the next step.

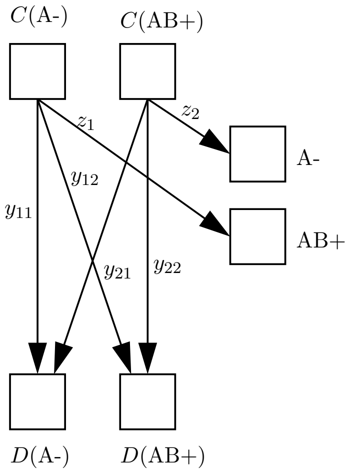

Figure B.3. The blood flow in blood inventory management.

To solve the greedy step using a linear program for any value function, we piece–wise linear features as described in Section 10.2:

$$\phi_k(s) = \left[u_k^\mathsf{T} k(s) + t_k\right]_+ .$$

In this problem, the function k is essentially an identity function. Assume that a value function is $v = \Phi x$. The linear program then becomes:

$$\begin{aligned} \max_{y,z} \quad & \sum_{ij} r_{ij} y_{ij} + \sum_k l_k x_k \\ \text{s.t.} \quad & \sum_j y_{ij} + z_k \le C(i) \quad \forall i \in \mathcal{T} \\ & \sum_i y_{ij} \le D(j) \quad \forall j \in \mathcal{T} \\ & y_{ij}, z_i \ge 0 \quad \forall i, j \in \mathcal{T} \\ & l_k \ge u_k^\mathsf{T} z - t_k \\ & l_k \ge 0 \end{aligned}$$

The variable l_k here represents the value of the linear feature. It is easy to show correctness of this linear program as long as all x_i are *non-positive* except those for which $t_k = 0$.

The simplest method for generating constraints is to simply sample actions or include actions that are greedy with respect to some particular value function. It is also possible to include maximum-flow linear programs directly to approximate linear programs. In that case, the term $l_k x_k$ is not linear. To obtain a linear representation, we can consider the dual of the linear program. Let the value of the flow for a dual variable a be $g(a, v)$ for a value function v. Then, it is easy to show that the optimal solutions of the following two linear programs are identical:

$$\begin{aligned} \min_v \quad & c^\mathsf{T} v \\ \text{s.t.} \quad & v \ge \min_a g(a, v) \\ & v \in \mathcal{M} \end{aligned}$$

and

$$\begin{aligned} \min_{v,a} \quad & c^{\mathsf{T}} v \\ \text{s.t.} \quad & v \geq g(a,v) \\ & v \in \mathcal{M} \end{aligned}$$

B.3 Reservoir Management

In this section, we formally describe the reservoir management problem. We also define general notation that can be used to represent other similar domains. In these problems, there is large uncertainty that is not under the control of the decision maker, but is instead external.

The state in the reservoir management is defined by the tuple (l, s, i, e). Here $l \in \mathbb{R}$ is the water level in the reservoir, $s \in \mathbb{Z}$ is the season of the year (such as the day or week of the year), $i \in \mathbb{R}^t$ is the forecast of the inflows, and $e \in \mathbb{R}^t$ is the forecast of the energy prices. The actions represent the discharge $d \in \mathbb{R}$ and are assumed to be discretized.

The transitions $(l, s, i, e) \to (l', s', i', e')$ is defined as follows:

$$\begin{aligned} l' &\leftarrow l + i(1) - d & s' &\leftarrow (s+1) \mod 365 \\ i(k) &\leftarrow i(k+1) & e(k) &\leftarrow e(k+1) \\ i(t) &\leftarrow Q_i(i) & e(t) &\leftarrow Q_e(e) \end{aligned}$$

Here, Q_i and Q_e are random variables distributed according to the model and t represents the last element.

To be able to generate samples, we used the available data to estimate stochastic models of the transitions. The data for inflows, desirable volume, and discharge comes from the Yellowtail River Dam in Montana. We used linear regression of the logarithm of the inflows.

The reward model is defined as follows.

$$r((l, s, i, e), d) = e(1)d - p_d(d) - p_l(l)$$

That is the monetary gain from generating electricity less the penalty for discharge outside of the desirable bounds $p_d : \mathbb{R} \to \mathbb{R}$ and the penalty for the water lever outside of the desirable bounds $p_l : \mathbb{R} \to \mathbb{R}$. The actual reward function we used in the experiments is as follows:

$$\begin{aligned} r((l, s, i, e), d) = {} & [\log(l) - 13.7]_+ + [13.6 - \log(l)]_+ + \\ & 0.03 * [\log(d) - 9.1]_+ + [6.8 - \log(d)]_+ + 0.0000001 * d * e, \end{aligned}$$

where e is the current price of electricity. Therefore, the controller ignores the volume and discharge of the reservoir as long as it is within reasonable boundaries. These boundaries were obtained to be centered about 95% of the values during the actual historical operation. This function is motivated by the operations of the reservoir, but does not precisely capture the tradeoffs.

The simplest state embedding function can be defined as:

$$k(l, s, i, e) = \big(l\big)\,.$$

That is, the states are mapped to a real space that represents the volume in the reservoir.

APPENDIX C

PROOFS

C.1 Basic Properties of Normed Vector Spaces

Definition C.1. A linear operator $Z : \mathbb{R}^n \to \mathbb{R}^m$ is monotonous if for all x and y:

$$x \geq y \Rightarrow Zx \geq Zy.$$

Lemma C.2. *For any vector x the following inequalities hold:*

$$\begin{aligned} \|x\|_s &\leq 2 \min_c \|x + c\mathbf{1}\|_\infty \leq 2\|x\|_\infty \\ \|x\|_\infty &\leq \|x\|_2 \\ \|x\|_2 &\leq \sqrt{n}\|x\|_1 \leq n\sqrt{n}\|x\|_\infty \\ \|x\|_{1,c} &\leq \|x\|_\infty, \end{aligned}$$

where n is the length of vector x and $\mathbf{1}^\mathsf{T} c = 1$ and $c \geq \mathbf{0}$.

Proof. The inequalities in the lemma are standard results of linear analysis. Because most proofs are straightforward, we only prove selected ones. Assume without loss of generality that $x \geq \mathbf{0}$, since for polynomial norms $\||x|\| = \|x\|$. Then from Jensen's inequality:

$$\|x\|_1^2 = n^2 \left(\sum_i \frac{x_i}{n} \right)^2 \geq n^2 \sum_i \frac{x_i^2}{n} = n \sum_i x_i^2 = n\|x\|_2^2.$$

□

C.2 Properties of the Bellman Operator

Lemma C.3. *The operator L_π is linear for any policy π.*

The lemma follows directly from the definitions.

Lemma C.4. *For any ϵ, and a value function v, the sets of greedy policies for v and $v + \epsilon\mathbf{1}$ are identical.*

The lemma follows directly from the definition of a greedy policy in Definition 2.8.

Lemma C.5 (Bellman Operator)**.** *For any value function v we have:*

$$L(v + \epsilon\mathbf{1}) = Lv + \gamma\epsilon\mathbf{1}.$$

Proof. Let π be the greedy policy with respect to $v + \epsilon\mathbf{1}$. Let P and r be the corresponding policy and rewards. Because P is a stochastic matrix, $P\mathbf{1} = \mathbf{1}$. We have:

$$\begin{aligned} L(v + \epsilon\mathbf{1}) &= \gamma P(v + \epsilon\mathbf{1}) + r \\ &= \gamma\epsilon\mathbf{1} + \gamma P v + r \\ &= \gamma\epsilon\mathbf{1} + Lv. \end{aligned}$$

The last equality follows from Lemma C.4. □

Definition C.6. A matrix P is called stochastic when all its elements are non-negative and all its rows sum to 1.

Notice that the transition matrix P_π of an MDP is a stochastic matrix.

Lemma C.7 (Monotonicity)**.** *Let P be a stochastic matrix. Then both the linear operators P and $(\mathbf{I} - \gamma P)^{-1}$ are monotonous:*

$$\begin{aligned} &x \geq y \Rightarrow Px \geq Py \\ &x \geq y \Rightarrow (\mathbf{I} - \gamma P)^{-1} x \geq (\mathbf{I} - \gamma P)^{-1} y \end{aligned}$$

for all x and y.

Proof. Since P is a stochastic matrix, we have that $\gamma Px \geq \gamma Py$ when $x \geq y$. Using induction, the same can be shown for $(\gamma P)^i$ for any $i \geq 0$. Now

$$(\mathbf{I} - \gamma P)^{-1} x = \sum_{i=0}^{\infty} (\gamma P)^i x \geq \sum_{i=0}^{\infty} (\gamma P)^i y = (\mathbf{I} - \gamma P)^{-1} x.$$

□

Lemma C.8 (Monotonicity)**.** *Bellman operator is monotonous. That is:*

$$x \geq y \Rightarrow Lx \geq Ly.$$

Proof. Let $x \geq y$, and let ψ be the optimal policy for value function y. Then:

$$\begin{aligned} \max_{\pi \in \Pi}(r_\pi + \gamma P_\pi x) - \max_{\pi \in \Pi}(r_\pi + \gamma P_\pi y) &\geq r_\psi + \gamma P_\psi x - r_\psi + \gamma P_\psi y \\ &= L_\psi(x - y) \geq x - y \geq 0, \end{aligned}$$

where the last inequality follows from Lemma C.7. □

Theorem 2.7. *[(Bellman, 1957)] A value function v^* is optimal if and only if $v^* = Lv^*$. Moreover, v^* is unique and satisfies $v^* \geq v_\pi$.*

Proof. First, notice that for any value function v of a policy π, we have $Lv \geq v$, because the maximization may be pair wise, and $\gamma Pv + r = v$. Now we show that if the value function is optimal, then $v^* = Lv^*$. Let v^* be the optimal value function. For the sake of a contradiction, assume that

$$Lv^* = \gamma P v^* + r \geq v^* + x,$$

where $x \geq 0$. Then we have, from monotonicity if $(\mathbf{I} - \gamma P)^{-1}$:

$$\begin{aligned} \gamma P v^* + r &\geq v^* + x \\ r &\geq (\mathbf{I} - \gamma P) v^* + x \\ (\mathbf{I} - \gamma P)^{-1} r &\geq v^* + (\mathbf{I} - \gamma P)^{-1} x \\ v &\geq v^* + x, \end{aligned}$$

for some v. This is a contradiction with the optimality of v^*. In addition, this construction show that if for some policy $Lv > v$ then there exists a policy with a strictly dominant value function. Therefore, since the number of policies if finite, there must exists a policy such that $v^* = Lv^*$. We show the optimality of this policy next.

Now, we show that if $v^* = Lv^*$ then the value function is optimal. Let $Lv^* = v^*$ and assume there exists $v > v^*$. Let P and r be the transition matrix and the reward function associated with the value function v. Then:

$$\begin{aligned} \gamma P^* + r &\leq v^* \\ r &\leq (\mathbf{I} - \gamma P) v^* \\ (\mathbf{I} - \gamma P)^{-1} v &\leq v^* \\ v &\leq v^* \end{aligned}$$

Here, the fist inequality follows from the fact that L may be seen as a element-wise maximization. Then, the rest follows from monotonicity of $(\mathbf{I} - \gamma P)^{-1}$. This is a contradiction with the assumption that $v > v^*$. □

The Bellman operator is a contraction under the L_∞ norm and the span seminorm.

Theorem C.9 ((Bellman, 1957))**.** *For a Bellman operator L with a discount factor γ and arbitrary vectors v_1 and v_2, the following holds for any policy $\pi \in \Pi$:*

$$\begin{aligned} \|L_\pi v_1 - L_\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty \\ \|L v_1 - L v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty \\ \|L v_1 - L v_2\|_s \leq \gamma \|v_1 - v_2\|_s \end{aligned}$$

For a proof of this property, see for example Theorem 6.6.6 in (Puterman, 2005). The contraction may be stronger when the Bellman operator satisfies certain properties. The proofs are identical for span seminorm and L_∞ norm since the policy is invariant to addition of a constant.

The primal linear formulation of the MDP is (Puterman, 2005):

$$\begin{aligned} \min_v \quad & c^{\mathsf{T}} v \\ \text{s.t.} \quad & A v \geq b \end{aligned} \tag{C.1}$$

The dual of the linear formulation of the MDP (Puterman, 2005). The dual linear program is:

$$\begin{aligned} \max_u \quad & b^{\mathsf{T}} u \\ \text{s.t.} \quad & A^{\mathsf{T}} u = c \\ & u \geq \mathbf{0} \end{aligned} \tag{C.2}$$

Using the same notation as the approximate linear program formulation (ALP) with c set to α. The optimal solution of (C.2) corresponds to the optimal state visitation frequencies, as described in Section 2.2.

Lemma C.10. *Let u_π be the state-action visitation frequency of policy π. This value is feasible in* (C.2) *and:*

$$\mathbf{1}^{\mathsf{T}} u = \frac{1}{1 - \gamma}.$$

Proof. Let $u_a(s) = u_\pi(s, \pi(s,a))$ for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. The lemma follows as:

$$\begin{aligned}
\sum_{a \in \mathcal{A}} u_a^\mathsf{T}(\mathbf{I} - \gamma P_a) &= c^\mathsf{T} \\
\sum_{a \in \mathcal{A}} u_a^\mathsf{T}(\mathbf{I} - \gamma P_a)\mathbf{1} &= c^\mathsf{T}\mathbf{1} \\
(1-\gamma) \sum_{a \in \mathcal{A}} u_a^\mathsf{T}\mathbf{1} &= 1 \\
u^\mathsf{T}\mathbf{1} &= \frac{1}{1-\gamma}.
\end{aligned}$$

□

Lemma C.11. *Assume two MDPs $\Phi_1 = (\mathcal{S}, \mathcal{A}, P, r, \alpha)$ and $\Phi_2 = (\mathcal{S}, \mathcal{A}, P, r + c\mathbf{1}, \alpha)$. The optimal policies of these MDPs are the same for arbitrary value of c.*

Proof. The proof follows from the dual linear program formulation of an MDP (C.2). Then from Lemma C.10, we get:

$$\arg\min_{u \in U} (b + c\mathbf{1})^\mathsf{T} u = \arg\min_{u \in U} b^\mathsf{T} u + c\mathbf{1}^\mathsf{T} u = \arg\min_{u \in U} b^\mathsf{T} u,$$

where U represents the feasible set of (C.2). □

Lemma 2.13. *Transitive feasible value functions are an upper bound on the optimal value function. Assume an ϵ-transitive-feasible value function $v \in \mathcal{K}(\epsilon)$. Then:*

$$v \geq v^* - \frac{\epsilon}{1-\gamma}\mathbf{1}.$$

Proof. Let P^* and r^* be the transition matrix and the reward vector of the policy. Then, we have using Lemma C.7:

$$\begin{aligned}
v &\geq Lv - \epsilon\mathbf{1} \\
v &\geq \gamma P^* v + r^* - \epsilon\mathbf{1} \\
(\mathbf{I} - \gamma P^*)v &\geq r^* - \epsilon\mathbf{1} \\
v &\geq (I - \gamma P^*)^{-1} r^* - \frac{\epsilon}{1-\gamma}
\end{aligned}$$

□

Proposition 2.14. *The set $\mathcal{K}$ of transitive-feasible value functions is convex. That is for any $v_1, v_2 \in \mathcal{K}$ and any $\beta \in [0,1]$ also $\beta v_1 + (1-\beta)v_2 \in \mathcal{K}$.*

Proof. The set of transitive-feasible functions is defined to satisfy:

$$\mathcal{K} = \left\{ v \,\middle|\, v \geq \max_{\pi \in \Pi} L_\pi v \right\}.$$

An equivalent formulation is that:

$$\mathcal{K} = \left\{ v \,\middle|\, \min_{\pi \in \Pi} v - L_\pi v \geq 0 \right\} = \bigcap_{\pi \in \Pi} \left\{ v \,|\, v - L_\pi v \geq 0 \right\}.$$

An intersection of a finite number of convex sets is also a convex set. □

Lemma C.12. *Assume Assumption 2.21. Then there exists $v' \in \mathcal{M} \cap \mathcal{K}$ such that*

$$\|v' - v^*\|_{1,c} \leq \|v' - v^*\|_\infty \leq \frac{2}{1-\gamma} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty,$$

where $\mathbf{1}^\mathsf{T} c = 1$ and $c \geq \mathbf{0}$.

Proof. Let $v \in \mathcal{M}$ be such that $\|v - v^*\|_\infty \leq \epsilon$. Let $v' = v + \epsilon \frac{1+\gamma}{1-\gamma} \mathbf{1}$. Clearly:

$$\|v' - v^*\|_\infty \leq \|v - v^*\|_\infty + \epsilon \frac{1+\gamma}{1-\gamma} = \frac{\epsilon}{1-\gamma},$$

and $v' \in \mathcal{M}$ from Assumption 2.21. To show that $v' \in \mathcal{K}$, we have from Lemma C.8, Theorem 2.7, and Lemma C.5:

$$\begin{aligned} v^* - \left(\epsilon - \epsilon \frac{1+\gamma}{1-\gamma}\right) \mathbf{1} \quad &\leq v' \quad &\leq v^* + \left(\epsilon + \epsilon \frac{1+\gamma}{1-\gamma}\right) \mathbf{1} \\ v^* - \gamma \left(\epsilon - \epsilon \frac{1+\gamma}{1-\gamma}\right) \mathbf{1} \quad &\leq L v' \quad &\leq v^* + \gamma \left(\epsilon + \epsilon \frac{1+\gamma}{1-\gamma}\right) \mathbf{1} \end{aligned}$$

Putting these equations together shows that $v' - Lv' \geq 0$ and thus $v' \in \mathcal{K}$. The bound on the weighted L_1 norm follows from Lemma C.2. □

Lemma C.13. *Assume a fixed policy π. Let u^α be the state visitation frequencies for an initial distribution α. Then:*

$$u^{\mathbf{1}_s}(s) \geq u^\alpha(s)$$

for any α. Note that u here represents state visitation frequencies, not state–action visitation frequencies as it is used in the remainder of the thesis.

Proof. We have that:

$$\begin{aligned}
u^{\alpha}(s) &= \mathbf{1}_s^{\mathsf{T}}\left(\mathbf{I}-\gamma P\right)^{-1}\mathbf{1}_s \\
&= \sum_{k=0}^{\infty}\alpha^{\mathsf{T}}(\gamma P)^k\mathbf{1}_s \\
&= \alpha^{\mathsf{T}}\mathbf{1}_s + \gamma\alpha^{\mathsf{T}}P\sum_{k=0}^{\infty}(\gamma P)^k\mathbf{1}_s \\
&= \alpha^{\mathsf{T}}\mathbf{1}_s + \sum_{s'\in\mathcal{S}}\gamma\alpha^{\mathsf{T}}P\mathbf{1}_{s'}\mathbf{1}_{s'}^{\mathsf{T}}\sum_{k=0}^{\infty}(\gamma P)^k\mathbf{1}_s \\
&= \alpha^{\mathsf{T}}\mathbf{1}_s + \gamma\alpha^{\mathsf{T}}P\left(\mathbf{1}_s u^{\mathbf{1}_s}(s) + \sum_{s'\in\mathcal{S}\setminus\{s\}}\sum_{k=0}^{\infty}\mathbf{1}_{s'}\mathbf{1}_{s'}^{\mathsf{T}}(\gamma P)^k\mathbf{1}_s\right) \\
&= \alpha^{\mathsf{T}}\mathbf{1}_s + \gamma\alpha^{\mathsf{T}}P\mathbf{1}_{s'}\left(\mathbf{1}_s u^{\mathbf{1}_s}(s) + \mathbf{1}_{\mathcal{S}\setminus\{s\}}u^{\mathbf{1}_{\mathcal{S}\setminus\{s\}}}(s)\right).
\end{aligned}$$

The lemma then follows from noticing that $\gamma\alpha^{\mathsf{T}}P\mathbf{1}_{s'} < 1$ and inductively analyzing which term of $u^{\mathbf{1}_s}(s)$ or $u^{\mathbf{1}_{\mathcal{S}\setminus\{s\}}}(s)$ is greater. □

C.3 Value Functions and Policies

Lemma C.14. *The following holds for any value function v:*

$$\|v^* - v\|_\infty \le \frac{1}{1-\gamma}\|v - Lv\|_\infty.$$

Proof. The lemma follows directly from Theorem C.9 as:

$$\|v^* - v\|_\infty = \|v^* - Lv + Lv - v\|_\infty \le \|v^* - Lv\|_\infty + \|Lv - v\|_\infty \le \gamma\|v^* - v\|_\infty + \|Lv - v\|_\infty.$$

The lemma then follows by moving $\gamma\|v^* - v\|_\infty$ to the left-hand side. □

Proposition 2.9. *The policy π greedy for a value-function v satisfies $Lv = L_\pi v \ge L_{\pi'} v$ for all policies $\pi' \in \Pi$. In addition, the greedy policy with respect to the optimal value function v^* is an optimal policy.*

Proof. The first part of the proposition follows directly from the definition of a greedy policy. To show the second part, we have for all $\pi' \in \Pi$:

$$\begin{aligned} L_\pi v^* &= Lv^* = v^* \ge v_{\pi'} \\ \gamma P_\pi v^* + r_\pi &= v^* \\ v^* &= (\mathbf{I} - \gamma P_\pi)^{-1} r_\pi \ge v_\pi = (\mathbf{I} - \gamma P_{\pi'})^{-1} r_{\pi'}. \end{aligned}$$

That is the definition of an optimal policy. □

Theorem 2.15. *Let $\tilde{v}$ be the approximate value function, and v_π be a value function of an arbitrary policy π. Then:*

$$\begin{aligned} \|v^* - v_\pi\|_\infty &\le \frac{1}{1-\gamma}\|\tilde{v} - L_\pi \tilde{v}\|_\infty + \|\tilde{v} - v^*\|_\infty \\ \|v^* - v_\pi\|_\infty &\le \frac{2}{1-\gamma}\|\tilde{v} - L_\pi \tilde{v}\|_\infty \end{aligned}$$

Proof. The first inequality may be shown as follows:

$$\|v_\pi - v^*\|_\infty \le \|v_\pi - \tilde{v}\|_\infty + \|\tilde{v} - v^*\|_\infty$$

Now, using Theorem C.9:

$$\begin{aligned} \|v_\pi - \tilde{v}\|_\infty &= \|L_\pi v_\pi - L_\pi \tilde{v} + L_\pi \tilde{v} - \tilde{v}\|_\infty \\ &\le \|L_\pi v_\pi - L_\pi \tilde{v}\|_\infty + \|L_\pi \tilde{v} - \tilde{v}\|_\infty \\ &\le \gamma\|v_\pi - \tilde{v}\|_\infty + \|L_\pi \tilde{v} - \tilde{v}\|_\infty \\ \|v_\pi - \tilde{v}\|_\infty &\le \frac{1}{1-\gamma}\|L_\pi \tilde{v} - \tilde{v}\|_\infty. \end{aligned}$$

This gives us:

$$\|v_\pi - v^*\|_\infty \le \frac{1}{1-\gamma}\|L_\pi \tilde{v} - \tilde{v}\|_\infty + \|\tilde{v} - v^*\|_\infty.$$

The second inequality follows from Lemma C.14. □

Theorem 2.16. *[Robust Policy Loss] Let π be the policy greedy with respect to $\tilde{v}$. Then:*

$$\|v^* - v_\pi\|_\infty \leq \frac{2}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty.$$

In addition, if $\tilde{v} \in \mathcal{K}$, the policy loss is minimized for the greedy policy and:

$$\|v^* - v_\pi\|_\infty \leq \frac{1}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty.$$

Proof. The bound in Theorem 2.15 is minimized by the greedy policy from the definition of Definition 2.8:

$$v - Lv \geq v - L_{\pi'} v \geq 0.$$

The policy loss bound follows from Theorem 2.15 and:

$$\|v_\pi - v^*\|_\infty \leq \|v_\pi - \tilde{v}\|_\infty$$

since $\tilde{v} \geq v^*$ as Lemma 2.13 shows, ignoring the term $\|\tilde{v} - v^*\|_\infty$. □

Theorem 2.17. *[Expected Policy Loss] Let π be a greedy policy with respect to a value function $\tilde{v}$ and let the state-action visitation frequencies of π be bounded as $\underline{u} \leq u_\pi \leq \bar{u}$. Then:*

$$\begin{aligned}\|v^* - v_\pi\|_{1,\alpha} &= \alpha^\mathsf{T} v^* - \alpha^\mathsf{T} \tilde{v} + u_\pi^\mathsf{T} (\tilde{v} - L\tilde{v}) \\ &\leq \alpha^\mathsf{T} v^* - \alpha^\mathsf{T} \tilde{v} + \underline{u}^\mathsf{T} [\tilde{v} - L\tilde{v}]_- + \bar{u}^\mathsf{T} [\tilde{v} - L\tilde{v}]_+ .\end{aligned}$$

The state-visitation frequency u_π depends on the initial distribution α, unlike v^*. In addition, when $\tilde{v} \in \mathcal{K}$, the bound is:

$$\begin{aligned}\|v^* - v_\pi\|_{1,\alpha} &\leq -\|v^* - \tilde{v}\|_{1,\alpha} + \|\tilde{v} - L\tilde{v}\|_{1,\bar{u}} \\ \|v^* - v_\pi\|_{1,\alpha} &\leq -\|v^* - \tilde{v}\|_{1,\alpha} + \frac{1}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty\end{aligned}$$

Proof. The bound is derived as:

$$\begin{aligned}\alpha^\mathsf{T} v^* - \alpha^\mathsf{T} v_\pi &= \alpha^\mathsf{T} v^* - \alpha^\mathsf{T} v_\pi + (u_\pi^\mathsf{T}(\mathbf{I} - \gamma P_\pi) - \alpha^\mathsf{T})\tilde{v} \\ &= \alpha^\mathsf{T} v^* - r_\pi^\mathsf{T} u_\pi + (u_\pi^\mathsf{T}(\mathbf{I} - \gamma P_\pi) - \alpha^\mathsf{T})\tilde{v} \\ &= \alpha^\mathsf{T} v^* - r_\pi^\mathsf{T} u_\pi + u_\pi^\mathsf{T}(\mathbf{I} - \gamma P_\pi)\tilde{v} - \alpha^\mathsf{T}\tilde{v} \\ &= \alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} + u_\pi^\mathsf{T} ((\mathbf{I} - \gamma P_\pi)\tilde{v} - r_\pi) \\ &= \alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} + u_\pi^\mathsf{T} (\tilde{v} - L\tilde{v}) .\end{aligned}$$

We used the fact that $u_\pi^\mathsf{T}(\mathbf{I} - \gamma P_\pi) - \alpha^\mathsf{T} = \mathbf{0}$ from the definition of state-action visitation frequencies. The inequalities for $\tilde{v} \in \mathcal{K}$ follow from Lemma C.19, Lemma C.10, and the trivial version of the Holder's inequality:

$$\begin{aligned}\alpha^\mathsf{T} v^* - \alpha^\mathsf{T}\tilde{v} &= -\|v^* - \tilde{v}\|_{1,\alpha} \\ u_\pi^\mathsf{T} (\tilde{v} - L\tilde{v}) &\leq \|u_\pi\|_1 \left\|\tilde{v} - L\tilde{v}\right\|_\infty = \frac{1}{1-\gamma}\|\tilde{v} - L\tilde{v}\|_\infty\end{aligned}$$

□

Theorem 2.19. *[Expected Policy Loss] Let π be a greedy policy with respect to a value function $\tilde{v}$ and let the state-action visitation frequencies of π be bounded as $\underline{u} \leq u_\pi \leq \bar{u}$. Then:*

$$\|v^* - v_\pi\|_{1,\alpha} \leq \left(\bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T}\right)(\tilde{v} - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ ,$$

where $P^* = P_{\pi^*}$. The state-visitation frequency u_π depends on the initial distribution α, unlike v^*. In addition, when $\tilde{v} \in \mathcal{K}$, the bound can be simplified to:

$$\|v^* - v_\pi\|_{1,\alpha} \leq \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)(\tilde{v} - v^*)$$

This simplification is, however, looser.

Proof. The proof extends Theorem 2.17 as:

$$\begin{aligned} u_\pi^\mathsf{T}(\tilde{v} - L\tilde{v}) &\leq u_\pi^\mathsf{T}\left[\tilde{v} - L\tilde{v}\right]_+ \leq \bar{u}^\mathsf{T}\left[\tilde{v} - L\tilde{v}\right]_+ \\ &= \bar{u}^\mathsf{T}(\tilde{v} - L\tilde{v}) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ \\ &\leq \bar{u}^\mathsf{T}(\tilde{v} - L^*\tilde{v}) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ \\ &= \bar{u}^\mathsf{T}(\tilde{v} - L^*\tilde{v} + L^*v^* - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ \\ &= \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)(\tilde{v} - v^*) + \bar{u}^\mathsf{T}\left[L\tilde{v} - \tilde{v}\right]_+ , \end{aligned}$$

where $L^* = L_{\pi^*}$. □

A similar bound can be derived as follows.

Proposition C.15. *Assume a value function $\tilde{v}$ and let π be the greedy policy. Also let u^* be the* optimal *state-action visitation frequencies with bounds $\underline{u}^* \leq u^* \leq \bar{u}^*$ and $P^* = P_{\pi^*}$ and $r^* = r_{\pi^*}$. Then:*

$$\begin{aligned} \|v^* - v_\pi\|_{1,\alpha} &= (u^*)^\mathsf{T}(r^* - (\mathbf{I} - P^*)\tilde{v}) + \alpha^\mathsf{T}\tilde{v} - \alpha^\mathsf{T}v_\pi \\ &= (u^*)^\mathsf{T}(r^* - (\mathbf{I} - P^*)\tilde{v}) + \|\tilde{v} - v_\pi\|_{1,\alpha} \\ &\leq (\underline{u}^*)^\mathsf{T}\left[r^* - (\mathbf{I} - P^*)\tilde{v}\right]_- + (\bar{u}^*)^\mathsf{T}\left[r^* - (\mathbf{I} - P^*)\tilde{v}\right]_+ + \|\tilde{v} - v_\pi\|_{1,\alpha}. \end{aligned}$$

Notice that u^* depends on the initial distribution α, unlike v^*.

It is not clear yet whether this bounds can be used to find approximate value functions.

Proof. The bound can be derived as follows:

$$\begin{aligned} \alpha^\mathsf{T}v^* - \alpha^\mathsf{T}v_\pi &= \alpha^\mathsf{T}v^* - \alpha^\mathsf{T}v_\pi - ((u^*)^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T})\tilde{v} \\ &= (r^*)^\mathsf{T}u^* - \alpha^\mathsf{T}v_\pi - ((u^*)^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T})\tilde{v} \\ &= (r^*)^\mathsf{T}u^* - \alpha^\mathsf{T}v_\pi - (u^*)^\mathsf{T}(\mathbf{I} - \gamma P^*)\tilde{v} + \alpha^\mathsf{T}\tilde{v} \\ &= \alpha^\mathsf{T}\tilde{v} - \alpha^\mathsf{T}v_\pi + (u^*)^\mathsf{T}\left((\mathbf{I} - \gamma P^*)\tilde{v} - r^*\right). \end{aligned}$$

We used the fact that $(u^*)^\mathsf{T}(\mathbf{I} - \gamma P^*) - \alpha^\mathsf{T} = \mathbf{0}$ from the definition of state-action visitation frequencies. □

C.4 Iterative Value Function Approximation

Proposition 3.1. *The least-squares formulation of $\mathcal{Z}$ in (3.2) minimizes the L_2 norm of a projection of the Bellman residual. Let v be the least-squares solution, then it satisfies:*

$$v = \arg\min_{v \in \mathcal{M}} \|\Phi^\mathsf{T}(L_\pi v - v)\|_2.$$

Proof. Define $Z = \Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T}$ and let v be a solution of $\mathcal{Z}(\pi)$ for a policy π. It is easy to show that $v = \Phi x$ for some x and $Zv = v$. Then:

$$\begin{aligned} v = \mathcal{Z}(\pi) &= \left(\mathbf{I} - \gamma\Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T}P_\pi\right)^{-1}\Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T}r_\pi \\ &= \left(\mathbf{I} - \gamma Z P_\pi\right)^{-1} Z r_\pi \end{aligned}$$

Then, we have that:

$$\begin{aligned} \left(\mathbf{I} - \gamma Z P_\pi\right) v &= Z r_\pi \\ \left(Z - \gamma Z P_\pi\right) v &= Z r_\pi \\ Z\left(\mathbf{I} - \gamma P_\pi\right) v &= Z r_\pi \\ \Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T}\left(\mathbf{I} - \gamma P_\pi\right) v &= \Phi\left(\Phi^\mathsf{T}\Phi\right)^{-1}\Phi^\mathsf{T} r_\pi \\ \Phi^\mathsf{T}\left(\mathbf{I} - \gamma P_\pi\right) v &= \Phi^\mathsf{T} r_\pi \end{aligned}$$

Using that $\left(\Phi^\mathsf{T}\Phi\right)^{-1}$ is a regular matrix. The theorem then follows since $\Phi^\mathsf{T}\left(\mathbf{I} - \gamma P_\pi\right) v = \Phi^\mathsf{T} r_\pi$ is the optimality condition for $\min_{v \in \mathcal{M}} \|\Phi^\mathsf{T}(L_\pi v - v)\|_2$. □

Theorem 3.3. *Let $\hat{v}_k$ be the value function of the policy π_k in step k in L_∞-API. Then:*

$$\limsup_{k\to\infty} \|v^* - v_k\|_\infty \le \frac{2\gamma}{(1-\gamma)^3} \limsup_{k\to\infty} \min_{v \in \Phi} \|L_{\pi_k} v - v\|_\infty.$$

Proof. We use $L_k = L_{\pi_k}$ to simplify the notation. Let $\epsilon_k = \min_{v \in \mathcal{M}} \|L_{\pi_k} v - v\|_\infty$. From assumptions of the theorem:

$$\begin{aligned} -\epsilon_k \mathbf{1} \le &\quad \tilde{v}_k - L_k \tilde{v}_k &&\le \epsilon_k \mathbf{1} \\ -\epsilon_k \mathbf{1} \le &\quad (\mathbf{I} - L_k)\tilde{v}_k &&\le \epsilon_k \mathbf{1} \\ v_k - \epsilon_k\left(\mathbf{I} - \gamma P_k\right)^{-1}\mathbf{1} \le &\quad \tilde{v}_k &&\le v_k + \epsilon_k\left(\mathbf{I} - \gamma P_k\right)^{-1}\mathbf{1} \\ v_k - \tilde{\epsilon}_k \mathbf{1} \le &\quad \tilde{v}_k &&\le v_k + \tilde{\epsilon}_k \mathbf{1} \end{aligned}$$

Notice the definition of $\tilde{\epsilon}_k = 1/(1-\gamma)\epsilon_k$. The Bellman operators involved satisfy the following basic assumptions:

$$\begin{aligned} L_{k+1}\tilde{v}_k &\ge L_k \tilde{v}_k \\ v^* &\ge v_{k+1} \\ L_{k+1} v_{k+1} &= v_{k+1} \\ L v_k &\ge L_{k+1} v_k \end{aligned}$$

Then, using the facts above, Lemma C.7, and Lemma C.5, we have that:

$$\begin{aligned}
L_{k+1}v_k &\geq L_{k+1}(\tilde{v}_k - \tilde{\epsilon}_k\mathbf{1}) \\
&= L_{k+1}\tilde{v}_k - \gamma P_{k+1}\tilde{\epsilon}_k\mathbf{1} \\
&\geq L\tilde{v}_k - \gamma\tilde{\epsilon}_k\mathbf{1} \\
&\geq Lv_k - 2\gamma\tilde{\epsilon}_k\mathbf{1} \\
&\geq L_k v_k - 2\gamma\tilde{\epsilon}_k\mathbf{1} \\
&\geq v_k - 2\gamma\tilde{\epsilon}_k\mathbf{1}
\end{aligned}$$

$$\begin{aligned}
L_{k+1}v_k &\geq L_{k+1}(\tilde{v}_k - \epsilon_k \left(\mathbf{I} - \gamma P_k\right)^{-1}\mathbf{1}) \\
&= L_{k+1}\tilde{v}_k - \gamma P_{k+1}\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1} \\
&\geq L_k\tilde{v}_k - \gamma P_{k+1}\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1} \\
&\geq L_k v_k - \left(\gamma P_{k+1} + \gamma P_k\right)\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1} \\
&\geq v_k - \left(\gamma P_{k+1} + \gamma P_k\right)\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1}
\end{aligned}$$

Reformulating the last inequality above, the value function v_{k+1} is bounded as:

$$\begin{aligned}
\left(\gamma P_{k+1} + \gamma P_k\right)\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1} &\geq (\mathbf{I} - \gamma P_{k+1})v_k + r_k \\
v_{k+1} &\geq v_k - \left(\mathbf{I} - \gamma P_{k+1}\right)^{-1}\left(\gamma P_{k+1} + \gamma P_k\right)\left(\mathbf{I} - \gamma P_k\right)^{-1}\epsilon_k\mathbf{1} \\
&= Lv_k - \frac{2\gamma\tilde{\epsilon}_k}{1-\gamma}\mathbf{1}
\end{aligned}$$

To get a contraction, we need to bound the value v_{k+1} in terms of the Bellman operator. Using the inequality above and that $L_{k+1}v_k \geq Lv_k - 2\gamma\tilde{\epsilon}\mathbf{1}$, we get:

$$\begin{aligned}
v_{k+1} = L_{k+1}v_{k+1} &\geq L_{k+1}v_k - \frac{2\gamma^2\tilde{\epsilon}_k}{1-\gamma}\mathbf{1} \\
&\geq Lv_k - \frac{2\gamma^2\tilde{\epsilon}_k}{1-\gamma}\mathbf{1} - 2\gamma\tilde{\epsilon}_k \\
&= Lv_k - \frac{2\gamma\tilde{\epsilon}_k}{1-\gamma}\mathbf{1}
\end{aligned}$$

Plugging the inequality to the error bounds shows that the approximate policy update is truly a contraction offset by a constant:

$$\begin{aligned}
\|v^* - v_{k+1}\|_\infty &\leq \left\|v^* - Lv_k + \frac{2\gamma\tilde{\epsilon}_k}{1-\gamma}\mathbf{1}\right\|_\infty \\
&= \|Lv^* - Lv_k\|_\infty + \frac{2\gamma\tilde{\epsilon}_k}{1-\gamma} \\
&\leq \gamma\|v^* - v_k\|_\infty + \frac{2\gamma\tilde{\epsilon}_k}{1-\gamma}
\end{aligned}$$

The theorem then follows by taking the limit of both sides of the equation. □

Proposition 3.4. *Assume a policy $\pi \in \Pi$. There exists a constant $c \in \mathbb{R}$ such that for all $\gamma \in (0,1)$:*

$$\|v^* - v_\pi\|_\infty \le \frac{c}{1-\gamma},$$

There exists an MDP such that for some $c' \in \mathbb{R}$, which is independent of γ:

$$\frac{1}{(1-\gamma)^2}\|v_\pi - Lv_\pi\|_\infty \ge \frac{c'}{(1-\gamma)^3}.$$

Here, the left-hand side represents the bound from Theorem 3.2.

Proof. Let

$$\begin{aligned} v^*(\gamma) &= (\mathbf{I} - \gamma P^*)^{-1} r^* \\ v_\pi(\gamma) &= (\mathbf{I} - \gamma P_\pi)^{-1} r_\pi. \end{aligned}$$

Notice that P^* and r^* are also functions of the discount factor, but we omit the symbol to simplify notation. Consider, therefore, that the optimal policy π^* is chosen to maximize the bound:

$$\begin{aligned} \|v^*(\gamma) - v_\pi(\gamma)\|_\infty &= \|\left(\mathbf{I} - \gamma P^*\right)^{-1} r^* - \left(\mathbf{I} - \gamma P_\pi\right)^{-1} r_\pi\|_\infty \\ &\le \|\left(\mathbf{I} - \gamma P^*\right)^{-1} r^*\|_\infty + \|\left(\mathbf{I} - \gamma P_\pi\right)^{-1} r_\pi\|_\infty \\ &\le \frac{1}{1-\gamma}\max_{\pi^* \in \Pi} \|r^*\|_\infty + \frac{1}{1-\gamma}\|r_\pi\|_\infty \end{aligned}$$

Thus setting

$$c = \max_{\pi^* \in \Pi} \|r^*\|_\infty + \|r_\pi\|_\infty$$

proves the first part of the proposition.

The second part of the proposition can be shown using the following example. Consider an MDP with two states $\mathcal{S} = \{s_1, s_2\}$ and two actions $\mathcal{A} = \{a_1, a_2\}$ with the following transition matrices and rewards.

$$P_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$r_1 = \begin{pmatrix} 1 & 2 \end{pmatrix} \qquad r_2 = \begin{pmatrix} 2 & 2 \end{pmatrix}$$

Let policy π take action a_1 in all states, while the optimal policy has the transition matrix and reward P_2, r_2. □

Theorem 3.5. *Let $\tilde{v}$ be a solution of approximate policy or value iteration taken at any iteration. Then, there exists* no *constant c (independent of the representation $\mathcal{M}$) such that:*

$$\|v^* - \tilde{v}\|_\infty \le c \min_{v \in \mathcal{M}} \|v^* - v\|_\infty$$

$$\|\tilde{v} - L\tilde{v}\|_\infty \le c \min_{v \in \mathcal{M}} \|v - Lv\|_\infty$$

This result applies to approximate policy iteration with both L_2 Bellman residual and least-squares minimizations. The bounds also apply when the iterative algorithms converge.

The theorem follows from the following propositions.

Proposition C.16. *Let $v^* \in \mathcal{M}$. API may not converge to v^* when using* Bellman residual approximation*, assuming that it does not start with the optimal policy.*

Proof. This proof is based on a counterexample with three states and two policies π_1 and π_2. Let the current policy transition matrix and reward be P_1 and r_1. Let the optimal policy transition matrix be P_2 and reward r_2. These may be defined as:

$$P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \; r_1 = \begin{pmatrix} -5 \\ 1 \\ 2 \end{pmatrix} \; P_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \; r_2 = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix} .$$

Now, the corresponding values :

$$v_1 = \begin{pmatrix} 4 \\ 10 \\ 20 \end{pmatrix} \; {}_{\in} = \begin{pmatrix} 23 \\ 10 \\ 20 \end{pmatrix} .$$

The approximation basis then may be chosen as follows. Notice that the basis contains the optimal value function for policy π_2.

$$\Phi = \begin{pmatrix} 23 & 1 \\ 10 & 0 \\ 20 & 1 \end{pmatrix} .$$

Using approximate policy iteration as defined above and the basis M to approximate the value of the current policy using Bellman residual minimization, we get:

$$v_1 = M(B^T B)^{-1} B^T r_1 = \begin{pmatrix} 11.63 \\ 18.32 \\ 6.13 \end{pmatrix} ,$$

where $B = (\mathbf{I} - \gamma P_1)M$.

Therefore, policy π_1 is preferable to policy π_2, given the approximate value function. This is because for state s_1, the decision is based on

$$(\gamma v(s_2) + r_1) - (\gamma v(s_3) + r_2) = (-5 + 0.9 * 18.32) - (5 + 0.9 * 6.13) = 0.97.$$

Therefore, the algorithm does not find the optimal policy, though it is a part of the basis. □

Next, we show a similar property of least-squares approximation.

Proposition C.17. *Let $v^* \in \mathcal{M}$. Then API may not converge to v^* when using* least squares approximation*, assuming that it does not start with the optimal policy. This is also true for AVI.*

Proof. The proof is based on a counterexample with four states and two policies. Here, we use M to denote the basis. The two transition matrices are:

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad r_1 = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.6 \end{pmatrix} \quad P_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad r_2 = \begin{pmatrix} -1.0 \\ -1.0 \\ -1.0 \\ 0.6 \end{pmatrix}.$$

The true values of the two policies are:

$$v_1 = \begin{pmatrix} 1.00 \\ 1.00 \\ 1.00 \\ 6.00 \end{pmatrix} \quad v_2 = \begin{pmatrix} 1.66 \\ 2.96 \\ 4.40 \\ 6.00 \end{pmatrix},$$

Thus the optimal policy is π_2. The approximation basis is:

$$M = \begin{pmatrix} 1.66 & 3.00 \\ 2.96 & 2.90 \\ 4.40 & 0.00 \\ 6.00 & 0.00 \end{pmatrix}.$$

The approximate value $\tilde{v}$ for π_1 and the Q–values q_1 and q_2 are:

$$\tilde{v} = (\mathbf{I} - \gamma Z P_1)^{-1} Z r_1 = \begin{pmatrix} -0.20 \\ 0.04 \\ 0.77 \\ 1.06 \end{pmatrix} \quad q_1 = \begin{pmatrix} -0.08 \\ -0.08 \\ 0.14 \\ 1.55 \end{pmatrix} \quad q_2 = \begin{pmatrix} -0.96 \\ -0.30 \\ -0.05 \\ 1.55 \end{pmatrix},$$

where $Z = (M(M^T M)^{-1} M^T$. Vector q_1 dominates q_2 and therefore the algorithm does not converge to the optimal policy.

Value $\tilde{v}$ is also a fixed point of AVI, because π_1 is the optimal greedy policy and:

$$\begin{aligned} \tilde{v} &= (\mathbf{I} - \gamma Z P_1)^{-1} Z r_1 \\ \tilde{v} &= \gamma Z P_1 \tilde{v} + Z r_1. \end{aligned}$$

□

Notice that while the policy iteration does not converge to the optimal value, it is indicated by a large Bellman residual or approximation error.

Proposition 3.6. *The optimal value function v^* is a fixed point of approximate policy and value iteration whenever it is representable ($v^* \in \mathcal{M}$).*

Proof. The proposition follows from v^* being a fixed point with regard to the optimal greedy one-step update. We show the proposition in detail for least-squares approximation. First, let P be the transition matrix of a one-step optimal policy. Notice that this is in fact the optimal policy. We show that the value function in the next iteration is v^*. The value function satisfies:

$$(\mathbf{I} - \gamma Z P) v = Z r. \tag{C.3}$$

Equation (C.3) has a unique solution, because $(\mathbf{I} - \gamma Z P)$ is regular. This is because all eigenvalues of $\gamma Z P$ are less than 1 since Z is a projection matrix. Now $Z(\gamma P v^* + r) = Z v^* = v^*$, thus v^* is the solution of (C.3) and also the value function in the next iteration. □

Lemma C.18. *For any monotonous projection Z and transition matrix P, the matrix $(\mathbf{I} - \gamma Z P)^{-1}$ is also monotonous.*

The lemma follows from the fact that ZP is monotonous and $(\mathbf{I} - \gamma Z P)^{-1} = \sum_{i=0}^{\infty} (\gamma Z P)^i$.

Theorem 3.7. *Let Z be a* monotonous *(see Definition C.1) approximation operator and assume that either least-squares approximate policy or value iterations converge to $\tilde{v}$. Then we have:*

$$\begin{gathered}(\mathbf{I} - ZP^*)^{-1}(Z - \mathbf{I})v^* \leq \tilde{v} - v^* \leq (\mathbf{I} - Z\tilde{P})^{-1}(Z - \mathbf{I})\tilde{v}^* \\ \|\tilde{v} - v^*\|_\infty \leq \frac{1}{1-\gamma} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty,\end{gathered}$$

where P^* and $\tilde{P}$ are the transition matrices of policies greedy with regard to v^* and v respectively.

Proof. First, we show this for AVI. Both inequalities depend on monotonicity of Z. Let T^* and $\tilde{T}$ be the update operators greedy with regard to v^* and $\tilde{v}$ respectively.

$$\begin{aligned}\tilde{v} - v^* &= Z\tilde{T}\tilde{v} - T^* v^* \\ &= Z\tilde{T}\tilde{v} - Z\tilde{T}v^* + Z\tilde{T}v^* - T^*v^* \\ &\leq Z\tilde{T}\tilde{v} - Z\tilde{T}v^* + ZT^*v^* - T^*v^* \\ &= \gamma Z\tilde{P}(\tilde{v} - v^*) + (Z - \mathbf{I})v^*\end{aligned}$$

Now, we have using the monotonicity of $(\mathbf{I} - \gamma Z P)^{-1}$:

$$\tilde{v} - v^* \leq (\mathbf{I} - \gamma Z\tilde{P})^{-1}(Z - \mathbf{I})T^*v^*.$$

The opposite direction may be shown similarly.

$$\begin{aligned}\tilde{v} - v^* &= Z\tilde{T}\tilde{v} - T^* v^* \\ &= Z\tilde{T}\tilde{v} - ZT^*v^* + ZT^*v^* - T^*v^* \\ &\geq ZT^*\tilde{v} - ZT^*v^* + ZT^*v^* - T^*v^* \\ &= \gamma ZP^*(\tilde{v} - v^*) + (Z - \mathbf{I})T^*v^*\end{aligned}$$

From this, we have using Lemma C.18:

$$\tilde{v} - v^* \geq (\mathbf{I} - \gamma ZP^*)^{-1}(Z - \mathbf{I})v^*.$$

To show this for API, we need to show that if $\tilde{v}$ is a fixed point of approximate policy iteration, it is also a fixed point of value iteration. This is because $\tilde{v} = \gamma ZP\tilde{v} + Zr$.

$$\begin{aligned}\tilde{v} &= (\mathbf{I} - \gamma ZP)^{-1}Zr \\ (\mathbf{I} - \gamma ZP)\tilde{v} &= Zr \\ \tilde{v} &= \gamma ZP\tilde{v} + Zr.\end{aligned}$$

□

Theorem 3.9. *A linear approximation is monotonous if and only if it represents an aggregation.*

Proof. Because this theorem is not crucial in the work and its proof is quite technical, we only outline it here. The proof is based on analyzing the individual cases of the approximation feature properties.

Let x be the first column of Φ, y be the second one, and $y_1, y_2 \dots$ be the remaining ones. Assume without loss of generality that Φ is constructed so that:

$$\begin{aligned} x(1) &= 1 & x(n) &= 0 \\ y(1) &= 0 & y(n) &= 1 \\ y_i(1) &= 0 & y_i(n) &= 0, \end{aligned}$$

where n is the number of rows in Φ. Now consider the following options:

1. $x \geq 0 \quad \wedge \quad y \geq 0$
   (a) $\exists i \quad x^\mathsf{T} y_i \neq 0$: set all $x^\mathsf{T} y_j = 0$ and $x^\mathsf{T} y = 0$ and $y^\mathsf{T} y_j = 0$ using Gram-Schmidt orthogonalization. Proceed to 1(c) or 2(c), depending on the non-negativity of the vectors.
   (b) $\forall i \quad x^\mathsf{T} y_i = 0$ but $x^\mathsf{T} y > 0$: It is then easy to show that either y_i's span the difference between x and y or that the projection of $\begin{pmatrix}1 & 0 & \dots\end{pmatrix}^\mathsf{T}$ is not monotonous. When y_i's span the difference, the argument can be inductively applied to other vectors and rows showing that the operator is indeed an aggregation.
   (c) $x^\mathsf{T} y_i = 0$ and $x^\mathsf{T} y \leq 0$: Then $x = 0$ and $y = 0$, since they are non-negative.
2. $x(i) < 0$ for some i (x and y) are exchangeable.
   (a) $\exists i \quad x^\mathsf{T} y_i \neq 0$: subtract y_i from x. set all $x^\mathsf{T} y_j = 0$ and $x^\mathsf{T} y = 0$ and $y^\mathsf{T} y_j = 0$ using Gram-Schmidt orthogonalization. Proceed to 1(c) or 2(c), depending on the non-negativity of the vectors.
   (b) $\forall i \quad x^\mathsf{T} y_i = 0$ but $x^\mathsf{T} y > 0$: It is then easy to show that either y_i's span the difference between x and y or that the projection of $\begin{pmatrix}1 & 0 & \dots\end{pmatrix}^\mathsf{T}$ is not monotonous. When y_i's span the difference, the argument can be inductively applied to other vectors and rows showing that the operator is indeed an aggregation.
   (c) $x^\mathsf{T} y_i = 0$ and $x^\mathsf{T} y \leq 0$:
      i. $x^\mathsf{T} y = 0$: Easy to show, since the projection of $\begin{pmatrix}1 & 0 & \dots\end{pmatrix}^\mathsf{T}$ will have negative elements.
      ii. $x^\mathsf{T} y \neq 0$: Orthogonalize x and y and proceed with the appropriate case.

□

C.5 Approximate Linear Programming

Theorem 4.2. *[Offline Policy Loss] Assume Assumption 2.21 and let $\tilde{v}$ be the solution of the approximate linear program in* (ALP) *and π be the greedy policy. Then:*

$$\|v^* - v_\pi\|_{1,\alpha} \le \min_{v\in\mathcal{M}\cap\mathcal{K}} \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)(v - v^*) \le 2\bar{u}^\mathsf{T}\mathbf{1} \min_{v\in\mathcal{M}} \|v^* - v\|_\infty$$

when $c^\mathsf{T} = \bar{u}^\mathsf{T}(\mathbf{I} - \gamma P^*)$. In addition,

$$\|v^* - v_\pi\|_{1,\alpha} \le \min_{v\in\mathcal{M}\cap\mathcal{K}} \bar{u}^\mathsf{T}(v - v^*) \le \frac{2\bar{u}^\mathsf{T}\mathbf{1}}{1-\gamma} \min_{v\in\mathcal{M}} \|v^* - v\|_\infty,$$

when $c^\mathsf{T} = \bar{u}^\mathsf{T}$. Notice that the first bound is tighter by a factor $1-\gamma$.

Proof. The first inequalities of the proposition use the fact that $v \in \mathcal{K}$. Then $[Lv - v]_+ = \mathbf{0}$ can be used to reformulate the bounds in Theorem 2.19 and Remark 2.18. To show the L_∞ bound, let $\hat{v}$ be the minimizer of $\min_{v\in\mathcal{M}} \|v^* - v\|_\infty$ and let $\|v^* - \hat{v}\|_\infty = \epsilon$. Then:

$$\begin{aligned} v^* - \epsilon\mathbf{1} &\le \hat{v} &&\le v^* + \epsilon\mathbf{1} \\ v^* - \gamma\epsilon\mathbf{1} &\le L\hat{v} &&\le v^* + \gamma\epsilon\mathbf{1} \\ -(1-\gamma)\epsilon\mathbf{1} &\le \hat{v} - L\hat{v} &&\le (1-\gamma)\epsilon\mathbf{1}. \end{aligned}$$

That means that $\hat{v} \in \mathcal{K}((1-\gamma)\epsilon)$ and using Lemma C.5 we have that:

$$v' = v + \frac{\epsilon}{1-\gamma}\mathbf{1} \in \mathcal{K}.$$

Then v' is feasible in (ALP) and is therefore an upper bound on the solution. The proposition then follows by simple algebraic manipulation. □

Notice also that a similar bound can be derived with $c^\mathsf{T} = \left(\bar{u}^\mathsf{T}(\mathbf{I} - P^*) - \alpha^\mathsf{T}\right)$.

Theorem 4.4. *Assume Assumptions 2.21, 2.26, 2.27, 2.28 and let v_1, v_2, v_3 be the optimal solutions of* (ALP), (s–ALP), *and* (e–ALP) *respectively. Then, the following inequalities hold:*

$$\begin{aligned} \|v_1 - v^*\|_{1,c} &\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty \\ \|v_2 - v^*\|_{1,c} &\le \|v_1 - v^*\|_{1,c} + 2\frac{\epsilon_p(\psi)}{1-\gamma} \\ &\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty + 2\epsilon_c(\psi) + 2\frac{\epsilon_p(\psi)}{1-\gamma} \\ \|v_3 - v^*\|_{1,c} &\le \|v_1 - v^*\|_{1,c} + 2\epsilon_c(\psi) + \frac{3\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma} \\ &\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - v^*\|_\infty + 2\epsilon_c(\psi) + \frac{3\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma} \end{aligned}$$

The last bound can be tightened to $2\epsilon_s$ from $3\epsilon_s(\psi)$ when $\epsilon_c = 0$.

Proof. To simplify the notation, we omit ψ in the notation of ϵ in the proof.

Proof of $\|v_1 - v^*\|_{1,c}$:
Let v be the minimizer of $\min_{v \in \mathcal{M}} \|v - v^*\|_\infty$. Then from Lemma C.12, Lemma 2.13, and $\|x\|_{1,c} \le \|x\|_\infty$ there exists $v' \in \mathcal{M} \cap \mathcal{K}$ such that:

$$\|v' - v^*\|_\infty \le \frac{2}{1-\gamma} \|v - v^*\|_\infty$$

From Lemma 2.13, we have that:

$$\begin{aligned} \|v_1 - v^*\|_{1,c} &= c^\mathsf{T}(v_1 - v^*) = c^\mathsf{T} v_1 - c^\mathsf{T} v^* \le c^\mathsf{T} v' - c^\mathsf{T} v^* \\ &= \|v_1 - v^*\|_{1,c}. \end{aligned}$$

Proof of $\|v_2 - v^*\|_{1,c}$:
Let v_2' be the solution (s–ALP) but with $c^\mathsf{T} v$ as the objective function. From Lemma 2.13 we have:

$$\begin{aligned} v_2 &\ge v^* - \frac{\epsilon_p}{1-\gamma}\mathbf{1} \\ v_2' &\ge v^* - \frac{\epsilon_p}{1-\gamma}\mathbf{1}. \end{aligned}$$

The difference between v_2 and v_2' can be quantified as follows, using their same feasible sets and the fact that v_2 is minimal with respect to $\bar{c}$:

$$c^\mathsf{T} v_2 \le \bar{c}^\mathsf{T} v_2 + \epsilon_c \le \bar{c}^\mathsf{T} v_2' + \epsilon_c \le c^\mathsf{T} v_2' + 2\epsilon_c.$$

Since $\mathcal{K}(\epsilon_p) \supseteq \mathcal{K}$ we also have that $c^\mathsf{T} v_2' \le c^\mathsf{T} v_1$. Then, using that $v_1 \in \bar{\mathcal{K}}$ and $v_1 \ge v^*$:

$$\begin{aligned} \|v_2 - v^*\|_{1,c} &\le \left\| v_2 - v^* + \frac{\epsilon_p}{1-\gamma}\mathbf{1} - \frac{\epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} \\ &\le \left\| v_2 - v^* + \frac{\epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} + \frac{\epsilon_p}{1-\gamma} \\ &\le c^\mathsf{T} \left(v_2 - v^* + \frac{\epsilon_p}{1-\gamma}\mathbf{1} \right) + \frac{\epsilon_p}{1-\gamma} \\ &\le c^\mathsf{T} (v_2 - v^*) + 2\frac{\epsilon_p}{1-\gamma} \\ &\le c^\mathsf{T} \left(v_2' - v^* \right) + 2\epsilon_c + 2\frac{\epsilon_p}{1-\gamma} \\ &\le c^\mathsf{T} (v_1 - v^*) + 2\epsilon_c + 2\frac{\epsilon_p}{1-\gamma} \\ &\le \|v_1 - v^*\|_{1,c} + 2\epsilon_c + 2\frac{\epsilon_p}{1-\gamma} \end{aligned}$$

Proof of $\|v_3 - v^*\|_{1,c}$:
Let v_3' be the solution (e–ALP) but with c as the objective function. Using Lemma 2.13 we have:

$$\begin{aligned} v_3 &\ge \tilde{L} v \ge \bar{L} v - \epsilon_s \mathbf{1} \ge L v - (\epsilon_s + \epsilon_p)\mathbf{1} \\ v_3 &\ge v^* - \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \end{aligned}$$

All of these inequalities also hold for v_3' since it is also feasible in (e–ALP). From Lemma C.5:

$$v_1 + \frac{\epsilon_s}{1-\gamma}\mathbf{1} \in \tilde{\mathcal{K}}.$$

Therefore:

$$c^{\mathsf{T}} v_3' \le c^{\mathsf{T}} v_1 + \frac{\epsilon_s}{1-\gamma}$$

The difference between v_3 and v_3' can be bounded as follows, using their same feasible sets and the fact that v_3 is minimal with respect to $\bar{c}$:

$$c^{\mathsf{T}} v_3 \le \bar{c}^{\mathsf{T}} v_3 + \epsilon_c \le \bar{c}^{\mathsf{T}} v_3' + \epsilon_c \le c^{\mathsf{T}} v_3' + 2\epsilon_c$$

Then:

$$\begin{aligned}
\|v_3 - v^*\|_{1,c} &\le \left\| v_3 - v^* + \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} - \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} \\
&\le \left\| v_3 - v^* + \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} + \frac{\epsilon_s + \epsilon_p}{1-\gamma} \\
&\le c^{\mathsf{T}} \left(v_3 - v^* + \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \right) + \frac{\epsilon_s + \epsilon_p}{1-\gamma} \\
&\le c^{\mathsf{T}} (v_3 - v^*) + 2\frac{\epsilon_s + \epsilon_p}{1-\gamma} \\
&\le c^{\mathsf{T}} (v_3' - v^*) + 2\epsilon_c + 2\frac{\epsilon_s + \epsilon_p}{1-\gamma} \\
&\le c^{\mathsf{T}} (v_1 - v^*) + 2\epsilon_c + \frac{3\epsilon_s + 2\epsilon_p}{1-\gamma} \\
&\le \|v_1 - v^*\|_{1,c} + 2\epsilon_c + \frac{3\epsilon_s + 2\epsilon_p}{1-\gamma}
\end{aligned}$$

□

Theorem 4.5. *[Online Error Bound] Let $\tilde{v} \in \tilde{\mathcal{K}} \cap \mathcal{M}$ be an arbitrary feasible solution of the estimated ALP* (e–ALP)*. Then:*

$$\|v^* - \tilde{v}\|_{1,c} \le \tilde{c}^{\mathsf{T}} \tilde{v} - c^{\mathsf{T}} v^* + \frac{2\epsilon_s(\psi) + 2\epsilon_p(\psi)}{1-\gamma}$$

Proof. To simplify the notation, we omit ψ in the notation of ϵ in the proof. Using Lemma 2.13 we have:

$$\begin{aligned}
v_3 &\ge \tilde{L}v \ge \bar{L}v - \epsilon_s\mathbf{1} \ge Lv - (\epsilon_s + \epsilon_p)\mathbf{1} \\
v_3 &\ge v^* - \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1}.
\end{aligned}$$

Then, using the above:

$$\begin{aligned}\|v - v^*\|_{1,c} &= \left\| v^* - v + \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} - \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} \\ &\leq \left\| v - v^* + \frac{\epsilon_s + \epsilon_p}{1-\gamma}\mathbf{1} \right\|_{1,c} + \frac{\epsilon_s + \epsilon_p}{1-\gamma} \\ &= c^\mathsf{T} v - cv^* + 2\frac{\epsilon_s + \epsilon_p}{1-\gamma} \\ &= \tilde{c}^\mathsf{T} v - cv^* + \epsilon_c + 2\frac{\epsilon_s + \epsilon_p}{1-\gamma}\end{aligned}$$

□

Proposition 4.6. *For any $\tilde{v} \in \mathcal{K}$ there exists no constant $c \in \mathbb{R}$ such that:*

$$\bar{u}^\mathsf{T}(\tilde{v} - v^*) \leq c\bar{u}^\mathsf{T}(\tilde{v} - L\tilde{v}),$$

even when Assumption 2.21 is satisfied. This holds for the precise Bellman operator L, not assuming sampling. In addition, for all $\tilde{v} \in \mathcal{K}$:

$$(\tilde{v} - v^*) \geq (\tilde{v} - L\tilde{v}).$$

Proof. We show a simple example. Consider and MDP with two states s_1 and s_2 and a single action. The transitions and rewards are defined as:

$$\begin{aligned} P(s_1, s_2) &= 1 & \qquad P(s_2, s_2) &= 1 \\ r(s_1) &= 1 & r(s_2) &= (1-\gamma). \end{aligned}$$

The optimal value function is $v^* = \mathbf{1}$. There is only one feature $\phi = \mathbf{1}$. The optimal solution of (ALP) for $c^\mathsf{T}\mathbf{1} > 0$ is:

$$\tilde{v} = \frac{1}{1-\gamma}\mathbf{1}.$$

Now, for any $\bar{u}^\mathsf{T}\mathbf{1} = 1$:

$$\begin{aligned}\bar{u}^\mathsf{T}(\tilde{v} - v^*) &= \frac{1}{1-\gamma} \\ \bar{u}^\mathsf{T}(\tilde{v} - L\tilde{v}) &= \frac{1}{1-\gamma}\end{aligned}$$

which proves the first part of the proposition. The remainder of the proposition follows from Remark 2.18. □

Proposition 4.7. *There exists an MDP such that for the optimal solution $\tilde{v}$ of* (ALP) *we have that*

$$\|\tilde{v} - v^*\|_{1,\alpha} \geq \frac{2}{1-\gamma} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

This holds even when Assumption 2.21 is satisfied. In addition, if $c = u_\pi$ (which is unknown), the ALP bound on the policy loss is:

$$\|\tilde{v} - v^*\|_{1,u_\pi} \geq \frac{2}{(1-\gamma)^2} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

There also exists an MDP such that for the greedy policy π with respect to the ALP solution the policy loss is:

$$\|v^* - v_\pi\|_{1,\alpha} \geq \frac{2\gamma}{1-\gamma} \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

Proof. The proof of the proposition is simple. Consider an MDP with 2 states: $\mathcal{S} = \{s_1, s_2\}$ and 1 action: $\mathcal{A} = \{a_1\}$. The transition matrix and reward for the single action are defined as:

$$P = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad r = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

There is only one feature $\phi_1 = \mathbf{1}$. The proposition then follows directly from the optimal and approximate value functions for the MDP when

$$\alpha = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The last part of the proof can be shown by adding a state s_3 with transition probabilities and rewards:

$$\begin{aligned} P(s_3, a_1) &= s_2 & \qquad r(s_3, a_1) &= 1 \\ P(s_3, a_2) &= s_3 & \qquad r(s_3, a_1) &= 1 - \epsilon, \end{aligned}$$

for some $\epsilon \to 0$ and the initial distribution:

$$\alpha = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

□

Lemma 4.10. *Let $\tilde{v}$ be feasible in* (ALP–e). *Then the following holds:*

$$\begin{aligned} \tilde{v} &\geq L^t \tilde{v} \\ \tilde{v} &\geq v^*, \end{aligned}$$

where L^t represents t consecutive applications of the Bellman operator L. In addition, for any value function v such that $\|v - v^*\|_\infty \leq \epsilon$ there exists a v' feasible in (ALP–e) *defined as:*

$$v' = v + \frac{\epsilon}{1 - \gamma^t} \mathbf{1}.$$

Proof. The property $\tilde{v} \geq L^t \tilde{v}$ uses the fact that the constraints are defined for any sequence of actions, including the ones taken in the multiple applications of the Bellman operator L. The proof then follows identically to comparable properties of regular ALPs: Lemma C.5, Lemma C.12, and Lemma 2.13. □

Proposition 4.11. *Assume Assumption 2.21 and let $\tilde{v}$ be the solution of the t-step expanded approximate linear program* (ALP–e).

$$\|v^* - \tilde{v}\|_{1,c} \leq \frac{2}{1-\gamma^t} \min_{v \in \mathcal{M}} \|v^* - v\|_\infty.$$

This bound does not guarantee a reduction of the policy loss of the greedy policy in Theorem 4.2.

Proof. Since $v \in \mathcal{K}$, the proposition follows directly from the following facts.

$$\begin{aligned} u_l^* &= \mathbf{0} \\ [r^* - (\mathbf{I} - P^*)\tilde{v}]_+ &= \mathbf{0} \\ \arg\min_{v \in \mathcal{K}} \|v - v_\pi\|_{1,\alpha} &= \arg\min_{v \in \mathcal{K}} \alpha^\mathsf{T} v - \alpha^\mathsf{T} v_\pi = \arg\min_{v \in \mathcal{K}} \alpha^\mathsf{T} v \end{aligned}$$

□

Proposition 4.13. *Let v_1 be the* optimal *solution of* (4.4) *and let $\bar{v}_1$ be the optimal solution of* (4.5)*. Let λ_1 and λ_2 be the Lagrange multipliers that correspond to constraints $A_1 v \geq b_1$ and $A_2 v \geq b_2$ respectively. Then the bound on the improvement from expanding constraints $A_1 v \geq b_1$ is at most:*

$$\begin{aligned} \|v_1 - v^*\|_{1,c} - \|\bar{v}_1 - v^*\|_{1,c} &\leq \|\lambda_1^\mathsf{T} A_1\|_1 \|v_1 - v_2\|_\infty \\ &\leq \frac{\|\,[Av_1 - b_1]_+\,\|_\infty}{1-\gamma} \|\lambda_1^\mathsf{T} A_1\|_1. \end{aligned}$$

Proof. First, let v_2 be the optimal solution of:

$$\begin{aligned} \min_{x} \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & \bar{A}_1 v \geq \bar{b}_1 \end{aligned}$$

From the optimality of v_1, and in particular from the dual feasibility and complementary slackness we have:

$$\begin{aligned} c &= A_1^\mathsf{T} \lambda_1 + A_2^\mathsf{T} \lambda_2 \\ \lambda_2^\mathsf{T} A_2 v_1 &= \lambda_2^\mathsf{T} b_2 \end{aligned}$$

We also have that:

$$\lambda_2^\mathsf{T} A_2 v_1 - \lambda_2^\mathsf{T} A_2 v_2 = \lambda_2^\mathsf{T} b_2 - \lambda_2^\mathsf{T} A_2 v_2 = \lambda_2^\mathsf{T} (b_2 - A_2 v_2) \leq 0,$$

from the feasibility of v_2 and from $\lambda_2 \geq \mathbf{0}$. Using the equations above, Lemma C.19, and the trivial version of Holder inequality we have:

$$\begin{aligned}
&\|v_1 - v^*\|_{1,c} - \|\bar{v}_1 - v^*\|_{1,c} = \\
&\quad = \quad c^{\mathsf{T}}(v_1 - v^*) - c^{\mathsf{T}}(v_1 - v_*) \leq c^{\mathsf{T}}(v_1 - v_2) \\
&\quad = \quad (\lambda_1^{\mathsf{T}} A_1 + \lambda_2^{\mathsf{T}} A_2)(v_1 - v_2) \\
&\quad = \quad \lambda_1^{\mathsf{T}} A_1(v_1 - v_2) + \lambda_2^{\mathsf{T}} A_2(v_1 - v_2) \\
&\quad \leq \quad \lambda_1^{\mathsf{T}} A_1(v_1 - v_2) \\
&\quad \leq \quad \|\lambda_1^{\mathsf{T}} A_1\|_1 \|v_1 - v_2\|_\infty .
\end{aligned}$$

The proposition follows from the standard Bellman residual bound on the value function approximation. □

Theorem 4.14. *[Offline Policy Loss] Let $\tilde{v}$ be the solution of the approximate linear program in* (ALP-r) *and π be the greedy policy. Then:*

$$\begin{aligned}
\|v^* - v_\pi\|_{1,\alpha} &\leq \min_{v \in \mathcal{M}} \left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)(\tilde{v} - v^*) + \bar{u}^{\mathsf{T}} \left[L\tilde{v} - \tilde{v}\right]_+ \\
&\leq \left(\mathbf{1}^{\mathsf{T}}\bar{u}(1-\gamma) - 1 + 2(1+\gamma)\mathbf{1}^{\mathsf{T}}\bar{u}\right) \min_{v \in \mathcal{M}} \|v - v^*\|_\infty
\end{aligned}$$

when $c^{\mathsf{T}} = \bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*)$ and the second inequality holds when $u^{\mathsf{T}}(\mathbf{I} - \gamma P^*) \geq \alpha^{\mathsf{T}}$. Note that the bound does not require Assumption 2.21 and does not involve transitive-feasible functions.

Proof. The first inequality follows directly from the derivation of (ALP-r). To prove the second inequality, assume that

$$\epsilon = \min_{v \in \mathcal{M}} \|v - v^*\|_\infty$$

with $\tilde{v}$ being the minimizer. Then, we have:

$$\begin{aligned}
-v^* - \epsilon\mathbf{1} \leq &\quad -\tilde{v} &&\leq -v^* + \epsilon\mathbf{1} \\
v^* - \epsilon\gamma\mathbf{1} \leq &\quad L\tilde{v} &&\leq v^* + \epsilon\gamma\mathbf{1} \\
v^* - \epsilon(1+\gamma)\mathbf{1} \leq &\quad L\tilde{v} - \tilde{v} &&\leq v^* + \epsilon(1+\gamma)\mathbf{1}
\end{aligned}$$

Now, when $u^{\mathsf{T}}(\mathbf{I} - \gamma P^*) \geq \alpha^{\mathsf{T}}$, we have:

$$\begin{aligned}
v^* - \epsilon\mathbf{1} \leq &\quad \tilde{v} &&\leq v^* + \epsilon\mathbf{1} \\
\epsilon\mathbf{1} \leq &\quad \tilde{v} - v^* &&\leq \epsilon\mathbf{1} \\
\epsilon\left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)\mathbf{1} \leq &\quad \left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)(v - v^*) &&\leq \epsilon\left(\bar{u}^{\mathsf{T}}(\mathbf{I} - \gamma P^*) - \alpha^{\mathsf{T}}\right)\mathbf{1}.
\end{aligned}$$

□

Proposition 4.15. *Assume Assumption 2.21 and that*

$$d > \frac{\mathbf{1}^{\mathsf{T}} c}{1-\gamma}\mathbf{1}.$$

Then the sets of optimal solutions of (ALP) *and* (ALP-rx) *are identical.*

Proof. We show that an optimal solution of (ALP-rx) that satisfies the hypothesis must be transitive feasible. Then since the objective functions of (ALP) and (ALP-rx) are the same, also their optimal solutions must match.

The proof is by contradiction. Assume that there is an optimal solution v_1 of (ALP-rx) and that it violates the constraint associated with coefficient d_i by $\bar{\epsilon}$. Let $v_2 = v_1 + \epsilon \mathbf{1}$ where $0 < \epsilon \leq \frac{\bar{\epsilon}}{1-\gamma}$. Then:

$$\begin{aligned} c^\mathsf{T}(v_2 - v^*) + d^\mathsf{T} \left[Lv_2 - v_2\right]_+ &= (v_1 - v^*) + \epsilon c^\mathsf{T}\mathbf{1} + d^\mathsf{T} \left[Lv_2 - v_2\right]_+ \\ &\leq (v_1 - v^*) + \epsilon c^\mathsf{T}\mathbf{1} + d^\mathsf{T} \left[Lv_1 - v_1\right]_+ + (1-\gamma)\epsilon c^\mathsf{T}\mathbf{1} d_i \\ &< (v_1 - v^*) + \epsilon c^\mathsf{T}\mathbf{1} + d^\mathsf{T} \left[Lv_1 - v_1\right]_+ + \epsilon c^\mathsf{T}\mathbf{1} \end{aligned}$$

using Lemma C.5. This contradicts the assumption that v_1 is optimal. □

C.6 Approximate Bilinear Programs

Theorem 5.2. *Given Assumption 2.21, any optimal solution $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ of the approximate bilinear program (ABP–L_∞) satisfies:*

$$\begin{aligned}\tilde{\pi}^\mathsf{T} \tilde{\lambda} + \tilde{\lambda}' = \|L\tilde{v} - \tilde{v}\|_\infty &\leq \min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty \\ &\leq 2 \min_{v \in \mathcal{M}} \|Lv - v\|_\infty \\ &\leq 2(1+\gamma) \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.\end{aligned}$$

Moreover, there exists an optimal solution $\tilde{\pi}$ that is greedy with respect to $\tilde{v}$ for which the policy loss is bounded by:

$$\|v^* - v_{\tilde{\pi}}\|_\infty \leq \frac{2}{1-\gamma} \left(\min_{v \in \mathcal{M}} \|Lv - v\|_\infty \right).$$

Proof. Let $\bar{v}$ be a value function with the minimal $\|L\bar{v} - \bar{v}\|_\infty$ feasible in approximate bilinear program (ABP–L_∞), and let $\bar{\pi}$ be a greedy policy with respect to $\bar{v}$. Because $\bar{v} \geq v^*$, as Lemma C.19 shows, we get:

$$t = \|L\bar{v} - \bar{v}\|_\infty = \|L_{\bar{v}}\bar{v} - \bar{v}\|_\infty.$$

Let f^* be the optimal objective value of (ABP–L_∞). Because both $\bar{v}$ and $\bar{\pi}$ are feasible in (ABP–L_∞), we have that $f^* \leq t$. Now, assume that $\tilde{v}$ is an optimal solution of (ABP–L_∞) with an objective value $\tilde{f} = \|L\tilde{v} - \tilde{v}\|_\infty > t$. Then, from item 5.5, $\tilde{f} > t \geq f^*$, which contradicts the optimality of $\tilde{v}$.

To show that the optimal policy is deterministic and greedy, let π^* be the optimal policy. Then consider the state s for which $\tilde{\pi}$ does not define a deterministic greedy action. From the definition of greedy action $\bar{a}$:

$$(L_{\bar{a}}\tilde{v})(s) \leq (L_{\tilde{\pi}}\tilde{v})(s).$$

From the bilinear formulation (ABP–L_∞), it is easy to show that there is an optimal solution such that:

$$\begin{aligned}(L_a\tilde{v})(s) &\leq \tilde{\lambda}' + \tilde{\lambda}(s,a) \\ \tilde{\lambda}(s,\bar{a}) &\leq \tilde{\lambda}(s,a).\end{aligned}$$

Then setting $\tilde{\pi}(s,\bar{a}) = 1$ and all other action probabilities to 0, the difference in the objective value function:

$$\tilde{\lambda}(s,\bar{a}) - \sum_{a \in \mathcal{A}} \tilde{\lambda}(s,a) \leq 0.$$

Therefore, the objective function for the deterministic greedy policy does not increase. The remainder of the theorem follows directly from Proposition C.21, Proposition C.22, and Proposition C.23. The bounds on the policy loss then follow directly from Theorem 2.16. The last inequality can be easily established using the same derivation as Theorem 4.14. □

Lemma 5.3. *Let $v \in \mathcal{K}$ be a transitive-feasible value function and let π be a policy. Then:*

$$f_1(\pi, v) \geq \|v - L_\pi v\|_\infty,$$

with an equality for a deterministic *policy π.*

Proof. The dual of the linear program (5.1) is the following program.

$$\begin{aligned} \max_{x} \quad & x^\mathsf{T}(Av - b) \\ \text{s.t.} \quad & x \leq \pi \\ & \mathbf{1}^\mathsf{T} x = 1 \\ & x \geq \mathbf{0} \end{aligned} \tag{C.4}$$

Note that replacing $\mathbf{1}^\mathsf{T} x = 1$ by $\mathbf{1}^\mathsf{T} x \leq 1$ preserves the properties of the linear program and would add an additional constraint in (ABP–L_∞): $\lambda' \geq 0$.

First we show that $f_1(\pi, v) \geq \|L_\pi v - v\|_\infty$. Because v is feasible in the approximate bilinear program (ABP–L_∞), $Av - b \geq 0$ and $v \geq Lv$ from Lemma C.19. Let state s be the state in which $t = \|L_\pi v - v\|_\infty$ is achieved. That is:

$$t = v(s) - \sum_{a \in \mathcal{A}} \pi(s, a) \left(r(s, a) + \sum_{s' \in \mathcal{S}} \gamma P(s', s, a) v(s') \right).$$

Now let $x(s, a) = \pi(s, a)$ for all $a \in \mathcal{A}$. This is a feasible solution with value t, from the stochasticity of the policy and therefore a lower bound on the objective value.

To show the equality for a deterministic policy π, we show that $f_1(\pi, v) \leq \|Lv - v\|_\infty$, using that $\pi \in \{0, 1\}$. Then let x^* be an optimal solution of (C.4). Define the index of x^* with the largest objective value as:

$$i \in \arg \max_{\{i \,|\, x^*(i) > 0\}} (Av - b)(i).$$

Let solution $x'(i) = 1$ and $x'(j) = 0$ for $j \neq i$, which is feasible since $\pi(i) = 1$. In addition:

$$(Av - b)(i) = \|L_\pi v - v\|_\infty.$$

Now $(x^*)^\mathsf{T}(Av - b) \leq (x')^\mathsf{T}(Av - b) = \|L_\pi v - v\|_\infty$, from the fact that i is the index of the largest element of the objective function. □

Lemma 5.5. *Let $v \in \mathcal{M} \cap \mathcal{K}$ be a transitive-feasible value function. There exists an optimal solution $\tilde{\pi}$ of the linear program (5.3) such that:*

1. *$\tilde{\pi}$ represents a* deterministic *policy*
2. *$L_{\tilde{\pi}} v = Lv$*
3. *$\|L_{\tilde{\pi}} v - v\|_\infty = \min_{\pi \in \Pi} \|L_\pi v - v\|_\infty = \|Lv - v\|_\infty$*

Proof. The existence of an optimal π that corresponds to a deterministic policy follows from Lemma 5.4, the correspondence between policies and values π, and the existence of a deterministic greedy policy.

Since $v \in \mathcal{K}$, we have for some policy π that:

$$v \geq Lv = L_\pi v \geq L_{\tilde{\pi}} v.$$

Assuming that $L_{\tilde{\pi}} < Lv$ leads to a contradiction since π is also a feasible solution in the linear program (5.3) and:

$$\begin{aligned} v - L_{\tilde{\pi}} &> v - Lv \\ \|v - L_{\tilde{\pi}}\|_\infty &> \|v - Lv\|_\infty. \end{aligned}$$

This proves the lemma. □

Theorem 5.6. *Given Assumption 2.21, any optimal solution* $(\tilde{\pi}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ *of the approximate bilinear program* (ABP–L_1) *satisfies:*

$$\begin{aligned} \frac{1}{1-\gamma}\left(\tilde{\pi}^\mathsf{T}\tilde{\lambda} + \tilde{\lambda}'\right) - \alpha^\mathsf{T}\tilde{v} = \frac{1}{1-\gamma}\|L\tilde{v} - \tilde{v}\|_\infty - \alpha^\mathsf{T} v &\leq \min_{v \in \mathcal{K}\cap\mathcal{M}} \left(\frac{1}{1-\gamma}\|Lv - v\|_\infty - \alpha^\mathsf{T} v\right) \\ &\leq \min_{v \in \mathcal{M}} \left(\frac{1}{1-\gamma}\|Lv - v\|_\infty - \alpha^\mathsf{T} v\right) \end{aligned}$$

Moreover, there exists an optimal solution $\tilde{\pi}$ that is greedy with respect to $\tilde{v}$ for which the policy loss is bounded by:

$$\|v^* - v_{\tilde{\pi}}\|_{1,\alpha} \leq \frac{2}{1-\gamma}\left(\min_{v\in\mathcal{M}} \frac{1}{1-\gamma}\|Lv - v\|_\infty - \|v^* - v\|_{1,\alpha}\right).$$

Proof. The proof of the theorem is almost identical to the proof of Theorem 5.2 with two main differences. First, the objective function of (ABP–L_1) is insensitive to adding a constant to the value function:

$$\|(\tilde{v} + k\mathbf{1}) - L(v + k\mathbf{1})\|_\infty - \alpha^\mathsf{T}(v + k\mathbf{1}) = \|\tilde{v} - Lv\|_\infty - \alpha^\mathsf{T} v.$$

Hence the missing factor 2 when going from minimization over $\mathcal{K}\cap\mathcal{M}$ to minimization over $\mathcal{M}$. The second difference is in the derivation of the bound on the policy loss, which follows directly from Theorem 2.17. □

Lemma 5.9. *Let value function v be feasible in the bilinear program* (ABP–U)*, and let π be an arbitrary policy. Then:*

$$f_1(\pi, v) \geq \|L_\pi v - v\|_{1,\bar{u}},$$

with an equality for a deterministic *policy.*

Proof. The dual of the linear program (5.4) program is the following.

$$\begin{aligned} \max_x \quad & x^\mathsf{T}(Av - b) \\ \text{s.t.} \quad & x \leq U^\mathsf{T}\pi \\ & x \geq \mathbf{0} \end{aligned} \tag{C.5}$$

We have that $f_1(\pi, v) \geq \|L_\pi v - v\|_{1,\bar{u}}$ since $x = U^\mathsf{T}\pi$ is a feasible solution. To show the equality for a deterministic policy π, let x^* be an optimal solution of linear program (C.5). Since $Av \geq b$ and U is non-negative, an optimal solution satisfies $x = U^\mathsf{T}\pi$. The optimal value of the linear program thus corresponds to the definition of the weighted L_1 norm. □

Lemma 5.10. *The following inequalities hold for any representable and transitive–feasible value functions:*

$$\min_{v\in\mathcal{M}\cap\mathcal{K}} \left(\|v - Lv\|_{1,\bar{u}(v)} - \|v^* - v\|_{1,\alpha}\right) \le \max\{1, \gamma\bar{u}(v)^{\mathsf{T}}\mathbf{1}\} \min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - v^*\|_\infty$$
$$\le \left(2 + \gamma + \mathbf{1}^{\mathsf{T}}\bar{u}(v) - \frac{1}{1-\gamma}\right) \min_{v\in\mathcal{M}} \|v - v^*\|_\infty$$

Proof. To simplify our notation, we use $\bar{u}$ instead of $\bar{u}(v)$, although it is a function of v. First, assume that $v \in \mathcal{M} \cap \mathcal{K}$, and it is a minimizer of:

$$\epsilon = \min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - v^*\|_\infty.$$

Then:

$$\begin{array}{rcl}
v^* \le & v & \le v^* + \epsilon\mathbf{1} \\
-v^* \ge & -Lv & \ge -val^* - \epsilon\mathbf{1} \\
0 \le & v - Lv & \le \gamma\epsilon\mathbf{1} \\
0 \le & \bar{u}^{\mathsf{T}}(v - Lv) & \le \gamma\epsilon\mathbf{1}^{\mathsf{T}}\bar{u} \\
0 \le & v - v^* & \le \epsilon\mathbf{1} \\
0 \le & \alpha^{\mathsf{T}}(v - v^*) & \le \epsilon\mathbf{1}
\end{array}$$

Putting these inequalities together and realizing that v is a feasible solution in the left–hand side minimization proves the first inequality.

To show the second inequality, assume that $v \in \mathcal{M}$, and it is be a minimizer of:

$$\epsilon = \min_{v\in\mathcal{M}} \|v - v^*\|_\infty.$$

It is also easy to show that when $v \in \mathcal{K}$, then:

$$\|v - Lv\|_{1,\bar{u}(v)} - \|v^* - v\|_{1,\alpha} = \bar{u}(v)^{\mathsf{T}}(v - Lv) - \alpha^{\mathsf{T}}(v - v^*)$$

Using a similar derivation as above, we get that:

$$-(1+\gamma)\epsilon\mathbf{1}^{\mathsf{T}}\bar{u} - \epsilon \le \quad \bar{u}^{\mathsf{T}}(v - Lv) - \alpha^{\mathsf{T}}(v - v^*) \quad \le (1+\gamma)\epsilon\mathbf{1}^{\mathsf{T}}\bar{u} + \epsilon$$

To get a transitive–feasible value function, let $\hat{v} = v + \frac{\epsilon}{1-\gamma}$. Then:

$$0 \le \bar{u}^{\mathsf{T}}(\hat{v} - L\hat{v}) - \alpha^{\mathsf{T}}(\hat{v} - v^*) \le \left(\mathbf{1}^{\mathsf{T}}\bar{u} - \frac{1}{1-\gamma}\right) + \bar{u}^{\mathsf{T}}(v - Lv) - \alpha^{\mathsf{T}}(v - v^*)$$

Putting these inequalities together and realizing that $\hat{v}$ is a feasible solution in the left–hand side minimization proves the second inequality. □

Lemma 5.12. *Let $v \in \mathcal{K}$ be a transitive-feasible value function and let π be a policy and U be defined as in Remark 5.7. Then:*

$$f_1(\pi, v) \ge \|v - L_\pi v\|_{k,\bar{u}},$$

with an equality for a deterministic *policy π.*

Proof. The dual of the linear program (5.1) program is the following.

$$\begin{aligned} \max_{x} \quad & x^{\mathsf{T}}(Av-b) \\ \text{s.t.} \quad & x \le U^{\mathsf{T}}\pi \\ & \mathbf{1}^{\mathsf{T}}\left(U^{\mathsf{T}}\right)^{-1} x \le k \\ & x \ge \mathbf{0} \end{aligned} \tag{C.6}$$

First change the variables in the linear program to $x = U^{\mathsf{T}} z$ to get:

$$\begin{aligned} \max_{z} \quad & z^{\mathsf{T}}U(Av-b) \\ \text{s.t.} \quad & z \le \pi \\ & \mathbf{1}^{\mathsf{T}} z \le k \\ & z \ge \mathbf{0} \end{aligned} \tag{C.7}$$

using the fact that U is diagonal and positive.

The norm$\|L_\pi v - v\|_{k,c}$ can be expressed as the following linear program:

$$\begin{aligned} \max_{y} \quad & y^{\mathsf{T}}XU(Av-b) \\ \text{s.t.} \quad & y \le \mathbf{1} \\ & \mathbf{1}^{\mathsf{T}} y \le k \\ & y \ge \mathbf{0} \end{aligned} \tag{C.8}$$

Here, the matrix $X : |\mathcal{S}| \times |\mathcal{S}| \cdot |\mathcal{A}|$ selects the subsets of the Bellman residuals that correspond the policy as defined:

$$X(s,(s',a')) = \begin{cases} \pi(s',a') & \text{when } s = s' \\ 0 & \text{otherwise} \end{cases}.$$

It is easy to shows that $v - L_\pi v = X(Av - b)$. Note that $XU = UX$ from the definition of U.

Clearly, when $\pi \in \{0, 1\}$ is deterministic the linear programs (C.7) and (C.8) are identical. When the policy π is stochastic, assume an optimal solution y of (C.8) and let $z = X^{\mathsf{T}} y$. Then, z is feasible in (C.7) with the identical objective value, which shows the inequality. □

Lemma C.19. *A value function v satisfies $Av \ge b$ if an only if $v \ge Lv$. In addition, if v is feasible in* (ABP–L_∞)*, then $v \ge v^*$.*

Proof. The backward implication of the first part of the lemma follows directly from the definition. The forward implication follows by an existence of $\lambda = \mathbf{0}$, $\lambda' = \| [Av - r]_+ \|_\infty$, which satisfy the constraints. The constraints on π are independent and therefore can be satisfied independently. The second part of the lemma also holds in ALPs (de Farias, 2002) and is proven identically. □

The minimization $\min_{v\in\mathcal{M}} \|Lv - v\|_\infty$ for a policy π can be represented as the following linear program.

$$\begin{aligned} \min_{\phi,v} \quad & \phi \\ \text{s.t.} \quad & (\mathbf{I} - \gamma P_\pi)v + \mathbf{1}\phi \geq r_\pi \\ & -(\mathbf{I} - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{aligned} \tag{C.9}$$

Consider also the following linear program.

$$\begin{aligned} \min_{\phi,v} \quad & \phi \\ \text{s.t.} \quad & (\mathbf{I} - \gamma P_\pi)v \geq r_\pi \\ & -(\mathbf{I} - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{aligned} \tag{C.10}$$

Next we show that the optimal solutions of (C.9) and (C.10) are closely related.

Lemma C.20. *Assume Assumption 2.21 and a given policy π. Let ϕ_1, v_1 and ϕ_2, v_2 optimal solutions of linear programs (C.9) and (C.10) respectively. Define:*

$$\bar{v}_1 = v_1 + \frac{\phi_1}{1-\gamma} \qquad\qquad \bar{v}_2 = v_1 - \frac{\phi_2}{2(1-\gamma)}\mathbf{1}$$

Then:

1. $2\phi_1 = \phi_2$
2. *$\bar{v}_1$ is an optimal solution in (C.10).*
3. *$\bar{v}_2$ is an optimal solution in (C.9).*
4. *Greedy policies with respect to v_1 and $\bar{v}_1$ are identical.*
5. *Greedy policies with respect to v_2 and $\bar{v}_2$ are identical.*

Proof. Let $\bar{\phi}_1 = 2\phi_1$ and $\bar{\phi}_2 = \frac{\phi_2}{2}$. We first show $\bar{\phi}_1, \bar{v}_1$ is feasible in (C.10). It is representable since $\mathbf{1} \in \mathcal{M}$ and it is feasible by the following simple algebraic manipulation:

$$\begin{aligned} (\mathbf{I} - \gamma P_\pi)\bar{v}_1 &= (\mathbf{I} - \gamma P_\pi)v_1 + (\mathbf{I} - \gamma P_\pi)\frac{\phi_1}{1-\gamma}\mathbf{1} \\ &= (\mathbf{I} - \gamma P_\pi)v_1 + \phi_1\mathbf{1} \\ &\geq -\phi_1\mathbf{1} + r_\pi + \phi_1\mathbf{1} \\ &= r_\pi \end{aligned}$$

and

$$\begin{aligned} -(\mathbf{I} - \gamma P_\pi)\bar{v}_1 + \bar{\phi}_1\mathbf{1} &= -(\mathbf{I} - \gamma P_\pi)\bar{v}_1 + 2\phi_1\mathbf{1} \\ &= -(\mathbf{I} - \gamma P_\pi)v_1 - (\mathbf{I} - \gamma P_\pi)\frac{\phi_1}{1-\gamma}\mathbf{1} + 2\phi_1\mathbf{1} \\ &= -(\mathbf{I} - \gamma P_\pi)v_1 - \phi_1\mathbf{1} + 2\phi_1\mathbf{1} \\ &\geq -\phi_1\mathbf{1} - r_\pi + 2\phi_1\mathbf{1} \\ &= -r_\pi \end{aligned}$$

Next we show that $\bar{\phi}_2, \bar{v}_2$ is feasible in (C.9). This solution is representable, since $\mathbf{1} \in \mathcal{M}$, and it is feasible by the following simple algebraic manipulation:

$$\begin{aligned}(\mathbf{I} - \gamma P_\pi)\bar{v}_2 + \bar{\phi}_2\mathbf{1} &= (\mathbf{I} - \gamma P_\pi)v_2 - (\mathbf{I} - \gamma P_\pi)\frac{\phi_2}{2(1-\gamma)}\mathbf{1} + \frac{\phi_2}{2}\mathbf{1} \\ &= (\mathbf{I} - \gamma P_\pi)v_2 - \frac{\phi_2}{2}\mathbf{1} + \frac{1}{2}\bar{\phi}_2\mathbf{1} \\ &= (\mathbf{I} - \gamma P_\pi)v_2 \\ &\geq r_\pi\end{aligned}$$

and

$$\begin{aligned}-(\mathbf{I} - \gamma P_\pi)\bar{v}_2 + \bar{\phi}_2\mathbf{1} &= -(\mathbf{I} - \gamma P_\pi)\bar{v}_2 + \frac{\phi_2}{2}\mathbf{1} \\ &= -(\mathbf{I} - \gamma P_\pi)v_2 - (\mathbf{I} - \gamma P_\pi)\frac{\phi_2}{1-\gamma}\mathbf{1} + \frac{\phi_2}{2}\mathbf{1} \\ &= -(\mathbf{I} - \gamma P_\pi)v_2 + \phi_2\mathbf{1} \\ &\geq -r_\pi\end{aligned}$$

It is now easy to shows that $\bar{\phi}_1, \bar{v}_1$ is optimal by contradiction. Assume that there exists a solution $\phi_2 < \bar{\phi}_1$. But then:

$$2\bar{\phi}_2 \leq \phi_2 < \bar{\phi}_1 \leq 2\phi_1,$$

which is a contradiction with the optimality of ϕ_1. The optimality of $\bar{\phi}_2, \bar{v}_2$ can be shown similarly. □

Proposition C.21. *Assumption 2.21 implies that:*

$$\min_{v \in \mathcal{M} \cap \mathcal{K}} \|Lv - v\|_\infty \leq 2 \min_{v \in \mathcal{M}} \|Lv - v\|_\infty.$$

Proof. Let $\hat{v}$ be the minimizer of $\hat{\phi} = \min_{v \in \mathcal{M}} \|Lv - v\|_\infty$, and let $\hat{\pi}$ be a policy that is greedy with respect to $\hat{v}$. Define:

$$\tilde{v} = \hat{v} + \frac{\hat{\phi}}{1-\gamma}.$$

Then from Lemma C.20:

1. Value function $\tilde{v}$ is an optimal solution of (C.10): $\tilde{v} \geq L_\pi \tilde{v}$
2. Policy $\hat{\pi}$ is greedy with regard to $\tilde{v}$: $L_{\hat{\pi}}\tilde{v} \geq L\tilde{v}$
3. $\|L_\pi \tilde{v} - \tilde{v}\|_\infty = 2\hat{\phi}$

Then using a simple algebraic manipulation:

$$\tilde{v} \geq L_{\hat{\pi}}\tilde{v} = L\tilde{v}$$

and the proposition follows from Lemma C.19. □

Proposition C.22. *Let $\tilde{v}$ be a solution of the approximate bilinear program* (ABP–L_∞) *and let:*

$$v' = v - \frac{1/2}{(1-\gamma)}\|Lv - v\|_\infty \mathbf{1}.$$

Then:

1. $\|Lv' - v'\|_\infty = \frac{\|Lv-v\|_\infty}{2}$.
2. *Greedy policies with respect to v and v' are identical.*

The proposition follows directly from Lemma C.20.

Proposition C.23. *Assumption 2.21 implies that:*

$$\min_{v \in \mathcal{M}} \|Lv - v\|_\infty \le (1+\gamma) \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

Proof. Assume that $\hat{v}$ is the minimizer of $\min_{v\in\mathcal{M}} \|v - v^*\|_\infty \le \epsilon$. Then:

$$\begin{aligned} v^* - \epsilon\mathbf{1} \le &\quad v &&\le v^* + \epsilon\mathbf{1} \\ Lv^* - \gamma\epsilon\mathbf{1} \le &\quad Lv &&\le Lv^* + \gamma\epsilon\mathbf{1} \\ Lv^* - \gamma\epsilon\mathbf{1} - v \le &\quad Lv - v &&\le Lv^* + \gamma\epsilon\mathbf{1} - v \\ Lv^* - v^* - (1+\gamma)\epsilon\mathbf{1} \le &\quad Lv - v &&\le Lv^* - v^* + (1+\gamma)\epsilon\mathbf{1} \\ -(1+\gamma)\epsilon\mathbf{1} \le &\quad Lv - v &&\le (1+\gamma)\epsilon\mathbf{1}. \end{aligned}$$

□

Theorem 5.13. *Let the optimal solutions to the sampled and precise Bellman residual minimization problems be:*

$$v_1 \in \arg\min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - Lv\|_\infty \qquad v_2 \in \arg\min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - \hat{L}v\|_\infty \qquad v_3 \in \arg\min_{v\in\mathcal{M}\cap\mathcal{K}} \|v - \tilde{L}v\|_\infty$$

Value functions v_1, v_2, v_3 correspond to solutions of instances of robust approximate bilinear programs for the given samples. Also let $\hat{v}_i = v_{\pi_i}$, where π_i is greedy with respect to v_i. Then, given Assumptions 2.21, 2.26, and 2.28, the following holds:

$$\begin{aligned} \|v^* - \hat{v}_1\|_\infty &\le \frac{2}{1-\gamma} \min_{v\in\mathcal{M}} \|v - Lv\|_\infty \\ \|v^* - \hat{v}_2\|_\infty &\le \frac{2}{1-\gamma} \left(\min_{v\in\mathcal{M}} \|v - Lv\|_\infty + \epsilon_p \right) \\ \|v^* - \hat{v}_3\|_\infty &\le \frac{2}{1-\gamma} \left(\min_{v\in\mathcal{M}} \|v - Lv\|_\infty + \epsilon_p + 2\epsilon_s \right) \end{aligned}$$

Proof. We show bounds on $\|v_i - Lv_i\|_\infty$; the remainder of the theorem follows directly from Theorem 5.2. The second inequality follows from Assumption 2.26 and Lemma C.5, as follows:

$$\begin{aligned} v_2 - Lv_2 &\le v_2 - \bar{L}v_2 \\ &\le v_1 - \bar{L}v_1 \\ &\le v_1 - Lv_1 + \epsilon_p\mathbf{1} \end{aligned}$$

The second inequality follows from Assumptions 2.26, 2.28 and Lemma C.5, as follows:

$$\begin{aligned} v_3 - Lv_3 &\le v_2 - \bar{L}v_2 + \epsilon_p\mathbf{1} \\ &\le v_2 - \tilde{L}v_2 + \epsilon_s\mathbf{1} + \epsilon_p\mathbf{1} \\ &\le v_1 - \tilde{L}v_1 + \epsilon_s\mathbf{1} + \epsilon_p\mathbf{1} \\ &\le v_1 - Lv_1 + 2\epsilon_s\mathbf{1} + \epsilon_p\mathbf{1} \end{aligned}$$

Here, we use the fact that $v_i \ge Lv_i$ and that v_i's minimize the corresponding Bellman residuals. □

Proposition 5.15. *Let $\tilde{v}_1$ and $\tilde{v}_2$ be feasible value functions in* (ABP–L_∞)*. Then the value function*

$$\tilde{v}(s) = \min\{\tilde{v}_1(s), \tilde{v}_2(s)\}$$

is also feasible in bilinear program (ABP–L_∞)*. Therefore $\tilde{v} \geq v^*$ and*

$$\|v^* - \tilde{v}\|_\infty \leq \min\left\{\|v^* - \tilde{v}_1\|_\infty, \|v^* - \tilde{v}_2\|_\infty\right\}.$$

Proof. Consider a state s and action a. Then from transitive feasibility of the value functions $\tilde{v}_1$ and $\tilde{v}_2$ we have:

$$\begin{aligned}\tilde{v}_1(s) &\geq \gamma \sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_1(s') + r(s, a) \\ \tilde{v}_2(s) &\geq \gamma \sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_2(s') + r(s, a).\end{aligned}$$

From the convexity of the min operator we have that:

$$\min\left\{\sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_1, \sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_2(s')\right\} \geq \sum_{s' \in \mathcal{S}} P(s', a, a) \min\{\tilde{v}_1(s'), \tilde{v}_2)(s')\}.$$

Then the proposition follows by the following simple algebraic manipulation:

$$\begin{aligned}\tilde{v} = \min\{\tilde{v}_1(s), \tilde{v}_2(s)\} &\geq \gamma \min\left\{\sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_1, \sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}_2(s')\right\} + r(s, a) \\ &\geq \gamma \sum_{s' \in \mathcal{S}} P(s', a, a) \min\{\tilde{v}_1(s'), \tilde{v}_2)(s')\} + r(s, a) \\ &= \gamma \sum_{s' \in \mathcal{S}} P(s', a, a)\tilde{v}(s) + r(s, a).\end{aligned}$$

□

C.6.1 NP-Completeness

Proposition C.24 (e.g. (Mangasarian, 1995))**.** *A bilinear program can be solved in NP time.*

There is an optimal solution of the bilinear program such that the solutions of the individual linear programs are basic feasible. The set of all basic feasible solutions is finite, because the feasible regions of w, x and y, z are independent. The value of a basic feasible solution can be calculated in polynomial time.

Theorem 5.16. *Assume $0 < \gamma < 1$, and a given $\epsilon > 0$. Then it is NP-complete to determine:*

$$\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty < \epsilon \qquad \min_{v \in \mathcal{M}} \|Lv - v\|_\infty < \epsilon.$$

The problem remains NP-complete when Assumption 2.21 is satisfied. It is also NP-complete to determine:

$$\min_{v \in \mathcal{M}} \|Lv - v\|_\infty - \|v^* - v\|_{1,\alpha} < \epsilon \qquad \min_{v \in \mathcal{M}} \|Lv - v\|_{1,\bar{u}} - \|v^* - v\|_{1,\alpha} < \epsilon,$$

assuming that $\bar{u}$ is defined as in Remark 5.7.

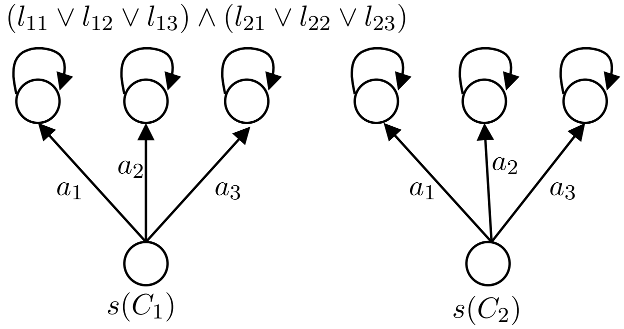
$(l_{11} \vee l_{12} \vee l_{13}) \wedge (l_{21} \vee l_{22} \vee l_{23})$
a_1
a_2
a_3
a_1
a_2
a_3
$s(C_1)$
$s(C_2)$

Figure C.1. MDP constructed from the corresponding SAT formula.

Proof. The membership in NP follows from Theorem 5.2 and Proposition C.24. We show NP-hardness by a reduction from the 3SAT problem. We first don't assume Assumption 2.21. We show the theorem for $\epsilon = 1$. The appropriate ϵ can be obtained by simply scaling the rewards in the MDP.

The main idea is to construct an MDP and an approximation basis, such that the approximation error is small whenever the SAT is satisfiable. The value of the states will correspond to the truth value of the literals and clauses. The approximation features ϕ will be used to constraint the values of literals that share the same variable. The MDP constructed from the SAT formula is depicted in Figure C.1.

Consider a SAT problem with clauses C_i:

$$\bigwedge_{i=1,\ldots,n} C_i = \bigwedge_{i=1,\ldots,n} (l_{i1} \vee l_{i2} \vee l_{i3}),$$

where l_{ij} are literals. A literal is a variable or a negation of a variable. The variables in the SAT are $x_1 \ldots x_m$. The corresponding MDP is constructed as follows. It has one state $s(l_{ij})$ for every literal l_{ij}, one state $s(C_i)$ for each clause C_i and an additional state $\bar{s}$. That is:

$$\mathcal{S} = \{s(C_i) \,|\, i = 1,\ldots,n\} \cup \{s(l_{ij}) \,|\, i = 1,\ldots,n, j = 1,\ldots,3\} \cup \{\bar{s}\}.$$

There are 3 actions available for each state $s(C_i)$, which determine the true literal of the clause. There is only a single action available in states $s(l_{ij})$ and $\bar{s}$. All transitions in the MDP are deterministic. The transition $t(s,a) = (s', r)$ is from the state s to s', when action a is taken, and the reward received is r. The transitions are the following:

$$t(s(C_i), a_j) = (s(l_{ij}), 1 - \gamma) \tag{C.11}$$

$$t(s(l_{ij}), a) = (s(l_{ij}), -(1 - \gamma)) \tag{C.12}$$

$$t(\bar{s}, a) = (\bar{s}, 2 - \gamma) \tag{C.13}$$

Notice that the rewards depend on the discount factor γ, for notational convenience.

There is one approximation feature for every variable x_k such that:

$$\begin{aligned}\phi_k(s(C_i)) &= 0\\ \phi_k(\bar{s}) &= 0\\ \phi_k(s(l_{ij})) &= \begin{cases}1 & \text{if } l_{ij} = x_k\\ -1 & \text{if } l_{ij} = \neg x_k\end{cases}\end{aligned}$$

An additional feature in the problem $\bar{\phi}$ is defined as:

$$\begin{aligned}\bar{\phi}(s(C_i)) &= 1\\ \bar{\phi}(s(l_{ij})) &= 0\\ \bar{\phi}(\bar{s}) &= 1.\end{aligned}$$

The purpose of state $\bar{s}$ is to ensure that $v(s(c_i)) \geq 2 - \gamma$, as we assume in the remainder of the proof.

First, we show that if the SAT problem is satisfiable, then $\min_{v \in \mathcal{M} \cap \mathcal{K}} \|Lv - v\|_\infty < 1$. The value function $\tilde{v} \in \mathcal{K}$ is constructed as a linear sum of the features as: $v = \Phi y$, where $y = (y_1, \ldots, y_m, \bar{y})$. Here y_k corresponds to ϕ_k and $\bar{y}$ corresponds to $\bar{\phi}$. The coefficients y_k are constructed from the truth value of the variables as follows:

$$\begin{aligned}y_k &= \begin{cases}\gamma & \text{if } x_k = \text{true}\\ -\gamma & \text{if } x_k = \text{false}\end{cases}\\ \bar{y} &= 2 - \gamma.\end{aligned}$$

Now define the *deterministic* policy π as:

$$\pi(s(C_i)) = a_j \text{ where } l_{ij} = \text{true}\,.$$

The true literals are guaranteed to exist from the satisfiability. This policy is greedy with respect to $\tilde{v}$ and satisfies that $\|L_\pi \tilde{v} - \tilde{v}\|_\infty \leq 1 - \gamma^2$.

The Bellman residuals for all actions and states, given a value function v, are defined as:

$$v(s) - \gamma v(s') - r,$$

where $t(s, a) = (s', r)$. Given the value function $\tilde{v}$, the residual values are:

$$\begin{aligned}t(s(C_i), a_j) = (s(l_{ij}), 1 - \gamma) : &\qquad \begin{cases}2 - \gamma - \gamma^2 + (1 - \gamma) = 1 - \gamma^2 & \text{if} \quad l_{ij} = \text{true}\\ 2 - \gamma + \gamma^2 + (1 - \gamma) = 1 + \gamma^2 & \text{if} \quad l_{ij} = \text{false}\end{cases}\\ t(s(l_{ij}), a) = (s(l_{ij}), (1 - \gamma)) : &\qquad \begin{cases}\gamma - \gamma^2 + 1 - \gamma = 1 - \gamma^2 & \text{if } l_{ij} = \text{true}\\ -\gamma + \gamma^2 + 1 - \gamma = (1 - \gamma)^2 > 0 & \text{if } l_{ij} = \text{false}\end{cases}\\ t(\bar{s}, a) = (\bar{s}, 1 - \gamma) : &\qquad (1 - \gamma) + \gamma - 1 = 0\end{aligned}$$

It is now clear that π is greedy and that:

$$\|L\tilde{v} - \tilde{v}\|_\infty = 1 - \gamma^2 < 1.$$

We now show that if the SAT problem is not satisfiable then $\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty \geq 1 - \frac{\gamma^2}{2}$. Now, given a value function v, there are two possible cases for each $v(s(l_{ij}))$: 1) a positive value, 2) a non-positive value. Two literals that share the same variable will have the same sign, since there is only one feature per each variable.

Assume now that there is a value function $\tilde{v}$. There are two possible cases we analyze: 1) all transitions of a greedy policy are to states with positive value, and 2) there is at least one transition to a state with a non-positive value. In the first case, we have that

$$\forall i \, \exists j, \; \tilde{v}(s(l_{ij})) > 0.$$

That is, there is a function $q(i)$, which returns the positive literal for the clause j. Now, create a satisfiable assignment of the SAT as follows:

$$x_k = \begin{cases} \text{true} & \text{if } l_{iq(i)} = x_k \\ \text{false} & \text{if } l_{iq(i)} = \neg x_k \end{cases},$$

with other variables assigned arbitrary values. Given this assignment, all literals with states that have a positive value will be also positive. Since every clause contains at least one positive literal, the SAT is satisfiable, which is a contradiction with the assumption. Therefore, there is at least one transition to a state with a non-positive value.

Let C_1 represent the clause with a transition to a literal l_{11} with a non-positive value, without loss of generality. The Bellman residuals at the transitions from these states will be:

$$b_1 = \tilde{v}(s(l_{11})) - \gamma\tilde{v}(s(l_{11})) + (1-\gamma) \geq 0 - 0 + (1-\gamma) = 1 - \gamma$$
$$b_1 = \tilde{v}(s(C_1)) - \gamma\tilde{v}(s(l_{11})) - (1-\gamma) \geq 2 - \gamma - 0 - 1 + \gamma = 1$$

Therefore, the Bellman residual $\tilde{v}$ is bounded as:

$$\|L\tilde{v} - \tilde{v}\|_\infty \geq \max\{b_1, b_2\} \geq 1.$$

Since we did not make any assumptions on $\tilde{v}$, the claim holds for all representable and transitive-feasible value functions. Therefore, $\min_{v \in \mathcal{M} \cap \mathcal{K}} \|Lv - v\|_\infty \leq 1 - \gamma^2$ is and only if the 3-SAT problem is feasible.

It remains to show that the problem remains NP-complete even when Assumption 2.21 holds. Define a new state $\bar{s}_1$ with the following transition:

$$t(\bar{s}_2, a) = (\bar{s}_2, -\frac{\gamma}{2}).$$

All previously introduced features ϕ are zero on the new state. That is $\phi_k(\bar{s}_1) = \bar{\phi}(\bar{s}_1) = 0$. The new constant feature is: $\hat{\phi}(s) = 1$ for all states $s \in \mathcal{S}$, and the matching coefficient is denoted as $\bar{y}_1$. When the formula is satisfiable, then clearly $\min_{v \in \mathcal{M} \cap \mathcal{K}} \|Lv - v\|_\infty \leq 1 - \gamma^2$ since the basis is now richer and the Bellman error on the new transition is less than $1 - \gamma^2$ when $\bar{y}_1 = 0$.

Now we show that when the formula is not satisfiable, then:

$$\min_{v \in \mathcal{M} \cap \mathcal{K}} \|Lv - v\|_\infty \geq 1 - \frac{\gamma^2}{2}.$$

This can be scaled to an appropriate ϵ by scaling the rewards. Notice that

$$0 \leq \bar{y}_1 \leq \frac{\gamma}{2}.$$

When $\bar{y}_1 < 0$, the Bellman residual on transitions $s(C_i) \to s(l_{ij})$ may be decreased by increasing $\bar{y}_1$ while adjusting other coefficients to ensure that $v(s(C_i)) = 2 - \gamma$. When

$\bar{y}_1 > \frac{\gamma}{2}$ then the Bellman residual from the state $\bar{s}_1$ is greater than $1 - \frac{\gamma^2}{2}$. Given the bounds on $\bar{y}_1$, the argument for $y_k = 0$ holds and the minimal Bellman residual is achieved when:

$$\begin{aligned} v(s(C_i)) - \gamma v(s(l_{ij})) - (1-\gamma) &= v(s(\bar{s}_1)) - \gamma v(s(\bar{s}_1)) + \frac{\gamma}{2} \\ 2 - \gamma - \gamma \bar{y}_1 - (1-\gamma) &= \bar{y}_1 - \gamma \bar{y}_1 + \frac{\gamma}{2} \\ \bar{y}_1 &= \frac{\gamma}{2}. \end{aligned}$$

Therefore, when the SAT is unsatisfiable, the Bellman residual is at least $1 - \frac{\gamma^2}{2}$.

The NP-completeness of $\min_{v \in \mathcal{M}} \|Lv - v\|_\infty < \epsilon$ follows trivially from Proposition C.21. The proof for $\|v - Lv\|_\infty - \alpha^\mathsf{T} v$ is almost identical. The difference is a new state $\hat{s}$, such that $\phi(\hat{s}) = \mathbf{1}$ and $\alpha(\hat{s}) = 1$. In that case $\alpha^\mathsf{T} v = 1$ for all $v \in \mathcal{M}$. The additional term thus has no effect on the optimization.

The proof can be similarly extended to the minimization of $\|v - Lv\|_{1,\bar{u}}$. Define $\bar{u}(C_i) = 1/n$ and $\bar{u}(l_{ij}) = 0$. Then the SAT problem is satisfiable if an only if $\|v - Lv\|_{1,\bar{u}} = 1 - \gamma^2$. Note that $\bar{u}$, as defined above, is not an upper bound on the visitation frequencies u_π. It is likely that the proof could be extended to cover the case $\bar{u} \geq u_\pi$ by more carefully designing the transitions from C_i. In particular, there needs to be high probability of returning to C_i and $\bar{u}(l_{ij} > 0$. □

C.7 Homotopy Methods for Solving Linear Programs

Proposition 6.5. *Given Assumption 6.4, the function $x(\psi)$ is piecewise linear. Algorithm 7 then converges to the optimal solution in a finite number of steps.*

Proof. We only provide a sketch of the proof here since this method is quite standard and is not the focus of this work. The formal proof would be similar to the convergence of other homotopy methods, such as DASSO (James et al., 2009). The *dual* solution in Algorithm 7 guarantees that the solution can move for some $\epsilon > 0$. That is it guarantees it will not move against one of the constraints satisfied with equality. If there is an optimal solution such that the regularization constraint *may* become inactive for $\bar{\psi}$ (when $\min \lambda(\psi_i) > 0$), then it is also optimal for any value $\psi \geq \bar{\psi}$. □

Theorem 6.10. *algorithm 6.2 converges to the optimal solution in a finite number of steps.*

Proof. Need to show that the algorithm will make progress in every step. Clearly, the solution is uniquely determined by the active set of variables and the regularization coefficient ψ. Therefore there is a finite number of steps between non-linearity points such that they decrease ψ. In the remainder of the proof we universally assume that the regularization constraint is active, unless specified otherwise. To show that we show it is possible to move at least ϵ from a non-linearity point, we analyze the cases independently.

Variables become active/inactive Assume that the current set of active variables is $\mathcal{B}$, and variables $\mathcal{I}$ are added to get a new active set $\bar{\mathcal{B}}$:

$$\bar{\mathcal{B}} = \mathcal{B} \cup \mathcal{I}.$$

We need to show that $\Delta x_{\mathcal{I}} \geq \mathbf{0}$.

First, from the negative value of $\mu_{\mathcal{I}}$ have that:

$$\begin{aligned}
\Delta \mu_{\mathcal{I}} &\leq \mathbf{0} \\
\chi A_{\mathcal{IC}}^{\mathsf{T}} A_{\mathcal{BC}} \Delta x_{\mathcal{B}} + e_{\mathcal{I}} \Delta \lambda &\leq \mathbf{0} \\
-A_{\mathcal{IC}}^{\mathsf{T}} A_{\mathcal{BC}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} e_{\mathcal{B}} \Delta \lambda + e_{\mathcal{I}} \Delta \lambda &\leq \mathbf{0} \\
\left(-A_{\mathcal{IC}}^{\mathsf{T}} A_{\mathcal{BC}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} e_{\mathcal{B}} + e_{\mathcal{I}} \right) \Delta \lambda &\leq \mathbf{0} \\
A_{\mathcal{IC}}^{\mathsf{T}} A_{\mathcal{BC}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} e_{\mathcal{B}} - e_{\mathcal{I}} &\leq \mathbf{0} \\
A_{\mathcal{IC}}^{\mathsf{T}} A_{\mathcal{BC}} \left(A_{\mathcal{BC}}^{\mathsf{T}} A_{\mathcal{BC}} \right)^{-1} e_{\mathcal{B}} &\leq e_{\mathcal{I}} \qquad \text{(C.14)}
\end{aligned}$$

Here we used the fact that $\Delta \lambda \leq 0$ from (6.15).

In order to show that it is possible to move in the new direction $\Delta x_{\mathcal{I}}$, we must show that $\Delta x_{\mathcal{I}} \geq \mathbf{0}$. We have that:

$$A_{\bar{\mathcal{B}}\mathcal{C}} = \begin{pmatrix} A_{\mathcal{B}\mathcal{C}} & A_{\mathcal{I}\mathcal{C}} \end{pmatrix}$$

The new direction $\Delta x_{\bar{\mathcal{B}}}$ must satisfy the following equalities from (6.16):

$$\begin{aligned} \Delta x_{\bar{\mathcal{B}}} &= -\left(\chi A_{\bar{\mathcal{B}}\mathcal{C}}^{\mathsf{T}} A_{\bar{\mathcal{B}}\mathcal{C}}\right)^{-1} e_{\bar{\mathcal{B}}} \Delta\lambda \\ \Delta x_{\bar{\mathcal{B}}} &= \left(A_{\bar{\mathcal{B}}\mathcal{C}}^{\mathsf{T}} A_{\bar{\mathcal{B}}\mathcal{C}}\right)^{-1} e_{\bar{\mathcal{B}}} c \\ \left(\begin{pmatrix} A_{\mathcal{B}\mathcal{C}} & A_{\mathcal{I}\mathcal{C}} \end{pmatrix}^{\mathsf{T}} \begin{pmatrix} A_{\mathcal{B}\mathcal{C}} & A_{\mathcal{I}\mathcal{C}} \end{pmatrix}\right) \begin{pmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{I}} \end{pmatrix} &= \begin{pmatrix} e_{\mathcal{B}} \\ e_{\mathcal{I}} \end{pmatrix} c \\ \begin{pmatrix} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} & A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \\ A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} & A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \end{pmatrix} \begin{pmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{I}} \end{pmatrix} &= \begin{pmatrix} e_{\mathcal{B}} \\ e_{\mathcal{I}} \end{pmatrix} c, \end{aligned}$$

for some constant $c > 0$. That is:

$$A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \Delta x_{\mathcal{B}} + A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} = c e_{\mathcal{B}} \tag{C.15}$$

$$A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \Delta x_{\mathcal{B}} + A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} = c e_{\mathcal{I}} \tag{C.16}$$

Putting together (C.14), (C.15), and (C.16), we get:

$$\begin{aligned} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \Delta x_{\mathcal{B}} + A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &= c e_{\mathcal{B}} \\ A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \Delta x_{\mathcal{B}} + A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &= c A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} e_{\mathcal{B}} \\ A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \Delta x_{\mathcal{B}} + A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &< c e_{\mathcal{I}} \\ c e_{\mathcal{I}} - A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} + A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &< c e_{\mathcal{I}} \\ A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} \left(A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}}\right) A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &< A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} \\ A_{\mathcal{I}\mathcal{C}}^{\mathsf{T}} \left(\mathbf{I} - A_{\mathcal{B}\mathcal{C}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}}\right)^{-1} A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}}\right) A_{\mathcal{I}\mathcal{C}} \Delta x_{\mathcal{I}} &> 0 \end{aligned}$$

The proof that progress is made when the variables become inactive is very similar, and essentially corresponds to a reversed version of the proof above.

Constraints become active / inactive Assume that the current set of active constraints is $\mathcal{C}$ and constraints $\mathcal{J}$ are added to the new active set $\bar{\mathcal{C}}$:

$$\bar{\mathcal{C}} = \mathcal{C} \cup \mathcal{J},$$

and the new direction is $\Delta \bar{x}_{\mathcal{B}}$. That is:

$$A_{\mathcal{B}\bar{\mathcal{C}}} = \begin{pmatrix} A_{\mathcal{B}\mathcal{C}} \\ A_{\mathcal{B}\mathcal{J}} \end{pmatrix}$$

We need to show that:

$$A_{\mathcal{B}\mathcal{J}} \Delta \bar{x}_{\mathcal{B}} \leq \mathbf{0}.$$

That is the constraints that recently became active also stay active on the current linear segment. We have for the current update direction Δx that for some constant $c \geq 0$:

$$\begin{aligned} A_{\mathcal{B}\mathcal{J}} \Delta x_{\mathcal{B}} &< 0 \\ A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} e_{\mathcal{B}} c &< \mathbf{0} \\ A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} e_{\mathcal{B}} &\leq \mathbf{0} \end{aligned} \tag{C.17}$$

Then, we have for some $c \geq 0$ that:

$$\begin{aligned} \Delta \bar{x}_{\mathcal{B}} &= \left(A_{\mathcal{B}\bar{\mathcal{C}}}^{\mathsf{T}} A_{\mathcal{B}\bar{\mathcal{C}}} \right)^{-1} ec \\ \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} + A_{\mathcal{B}\mathcal{J}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{J}} \right) \Delta \bar{x}_{\mathcal{B}} &= ec \\ \left(I + \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} A_{\mathcal{B}\mathcal{J}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{J}} \right) \Delta \bar{x}_{\mathcal{B}} &= \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} ec \\ \left(A_{\mathcal{B}\mathcal{J}} + A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} A_{\mathcal{B}\mathcal{J}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{J}} \right) \Delta \bar{x}_{\mathcal{B}} &= A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} ec \\ \left(I + A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} A_{\mathcal{B}\mathcal{J}}^{\mathsf{T}} \right) A_{\mathcal{B}\mathcal{J}} \Delta \bar{x}_{\mathcal{B}} &= A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} ec \\ \left(I + A_{\mathcal{B}\mathcal{J}} \left(A_{\mathcal{B}\mathcal{C}}^{\mathsf{T}} A_{\mathcal{B}\mathcal{C}} \right)^{-1} A_{\mathcal{B}\mathcal{J}}^{\mathsf{T}} \right) A_{\mathcal{B}\mathcal{J}} \Delta \bar{x}_{\mathcal{B}} &< 0 \end{aligned}$$

The last step follows from (C.17). The proof that progress is made when the constraints become inactive is very similar, and essentially corresponds to a reversed version of the proof above. □

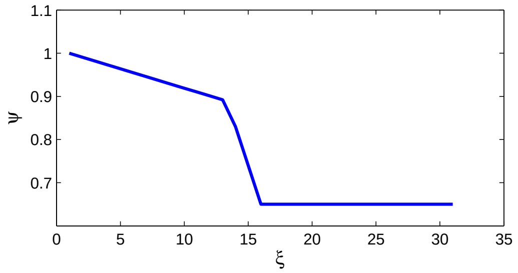

Figure C.2. Function θ_B for the counterexample.

C.8 Bilinear Program Solvers

Proposition 7.3. *The function $\theta_B(\psi)$ is not necessarily convex or concave.*

Proof. The proof is based on a simple counterexample. Consider an MDP with 4 states $s_1 \ldots s_4$. There are two actions with the following transition matrices:

$$P_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The rewards for the states are defined as:

$$r_1 = \begin{pmatrix} 1 \\ 1.5 \\ 1.5 \\ 2 \end{pmatrix} \qquad r_1 = \begin{pmatrix} 0 \\ 1.5 \\ 1.5 \\ 2 \end{pmatrix}.$$

The approximation features are defined as follows:

$$\Phi = \begin{pmatrix} 1 & 0 \\ 1 & 0.1 \\ 1 & 1 \\ 1 & 0. \end{pmatrix}$$

With the discount factor $\gamma = 0.95$, the function θ_B is neither convex, nor concave as Figure C.2.

□

Theorem 7.4. *Assume that for every state $s \in \mathcal{S}$ we have that $r(s, a_1) = r(s, a_2)$ for all $a_1, a_2 \in \mathcal{A}$. Also assume that $e(1) = 0$ and $e(i) > 0$ for all $i > 0$. For any $\bar{\psi}$ and $\psi \geq \bar{\psi}$ the function θ_B satisfies that:*

$$\theta_B(\psi) \geq \theta_B(0) - \frac{\psi}{\bar{\psi}}(\theta_B(0) - \theta_B(\bar{\psi})).$$

Proof. Because the rewards are independent of the action, we use $r(s) = r(s,a)$ for all actions. First, from Lemma 5.4, we have that:

$$\|v - Lv\|_\infty = \min_{\pi \in \Pi} \|\pi^\mathsf{T}(Av - b)\|_\infty.$$

When $\psi_0 = 0$, is is easy to show that $v = \mathbf{1}\tau$ and that:

$$\begin{aligned} v(s) - \gamma P(s,a_1)^\mathsf{T} v - r(s) &= \tau - \gamma\tau\mathbf{1} - r(s) \\ &= v(s) - \gamma P(s,a_2)^\mathsf{T} v - r(s) \end{aligned}$$

Now, assume without loss of generality that for some $v_1 \in \mathcal{M}(\psi_1)$:

$$v_1(s) - \gamma P(s,a_1)^\mathsf{T} v_1 + r(s) \le v_1(s) - \gamma P(s,a)^\mathsf{T} v_1 + r(s).$$

Then let $v_\beta = \beta v_0 + (1-\beta) v_1$ for $\beta \in [0,1]$ be a convex combination of v_0 and v_1. Also assume that:

$$v_1(s) - \gamma P(s,a_1)^\mathsf{T} v_1 \le v_1(s) - \gamma P(s,a_2)^\mathsf{T} v_1$$

for any $a_2 \in \mathcal{A}$. The Bellman residual of the value function v_β then satisfies that:

$$\begin{aligned} & v_\beta(s) - \gamma P(s,a_1)^\mathsf{T} v_\beta - r(s) = \\ &\quad = \beta(v_0(s) - \gamma P(s,a_1)^\mathsf{T} v_0 - r(s)) + (1-\beta)(v_1(s) - \gamma P(s,a_1)^\mathsf{T} v_1 - r(s)) \\ &\quad = \beta(v_0(s) - \gamma P(s,a_2)^\mathsf{T} v_0 - r(s)) + (1-\beta)(v_1(s) - \gamma P(s,a_1)^\mathsf{T} v_1 - r(s)) \\ &\quad \le \beta(v_0(s) - \gamma P(s,a_2)^\mathsf{T} v_0 - r(s)) + (1-\beta)(v_1(s) - \gamma P(s,a_2)^\mathsf{T} v_1 - r(s)) \\ &\quad = v_\beta(s) - \gamma P(s,a_2)^\mathsf{T} v_\beta - r(s). \end{aligned}$$

Therefore, there exists a policy π that is greedy with respect to v_β for all values of $\beta \in [0,1]$. It is also easy to show that $v_\beta \in \mathcal{M}((1-\beta)\psi_1) \cap \mathcal{K}$, which follows from the convexity of $\mathcal{K}$ (Proposition 2.14) and $\mathcal{M}$.

Now, define the following function:

$$g(\beta) = \|v_\beta - Lv_\beta\|_\infty = \max_{s \in \mathcal{S}} \mathbf{1}_s^\mathsf{T}(v - L_\pi v),$$

where π is the greedy policy defined above. The function g is convex, since it is a maximum of a set of linear functions; using linearity of L_π. This implies that:

$$\theta_B(\beta\psi_0 + (1-\beta)\psi_1) \le g(\beta).$$

To derive a contradiction, assume that there exists ψ_v that violates the inequality for from $\bar{\psi}$. Then:

$$\begin{aligned} \theta_B(\psi_v) &< \theta_B(0) - \frac{\psi_v}{\bar{\psi}}(\theta_B(0) - \theta_B(\bar{\psi})) \\ \frac{\psi_v}{\bar{\psi}}(\theta_B(\psi_v) - \theta_B(0) + \psi_v\theta_B(0)) &< \theta_B(\bar{\psi}) \\ (1-\beta)\theta_B(\psi_v) + \beta\theta_B(0) &< \theta_B(\bar{\psi}) \end{aligned}$$

Above, β is chosen so that $(1-\beta) = \frac{\psi_v}{\bar{\psi}}$. That mean that $\bar{\psi} = (1-\beta)\psi_v$ and using convexity of g:

$$\theta_B(\bar{\psi}) \le g(\bar{\psi}) \le (1-\beta)\theta_B(\psi_v) + \beta\theta_B(0) < \theta_B(\bar{\psi}),$$

which is a contradiction. □

Theorem 7.1. *Let $(\pi_1, \lambda_1, \lambda_1')$ be an optimal (greedy-policy) solution of* (ABP–L_∞). *Then:*

$$\left(\pi_1, \lambda_1, \lambda_1', z' = \min_{z \geq \lambda_1 - (\tau - \pi_1)} \mathbf{1}^\mathsf{T} z\right)$$

is an optimal solution of (ABP-MILP) *and vice versa, given that $\tau \geq \lambda_1$. When in addition f_1 and f_2 are the optimal objective values of* (ABP–L_∞) *and* (ABP-MILP)*, then $f_1 = f_2$.*

Proof. First, we show that $(\pi_1, \lambda_1, \lambda_1', z = \min_{z \geq \lambda_1 - (\tau - \pi_1)} \mathbf{1}^\mathsf{T} z)$ is feasible in (ABP-MILP) and has the same objective value. Since π_1 is a greedy policy (see Theorem 5.2), then $\pi_1 \in \{0, 1\}^{\mathcal{S} \times \mathcal{A}}$. That is π_1 is feasible in (ABP-MILP). Let then:

$$z_2(s,a) = \begin{cases} \lambda(s,a) & \text{if } \pi_1(s,a) = 1 \\ 0 & \text{otherwise} \end{cases} .$$

To show that z_2 is feasible in (ABP-MILP) analyze the following two cases:

1. $\pi(s,a) = 1$:
$$z_2(s,a) + \tau(s,a)(1 - \pi_1(s,a)) = z_2(s,a) = \lambda_1(s,a)$$
2. $\pi(s,a) = 0$:
$$z_2(s,a) + \tau(s,a)(1 - \pi_1(s,a)) \geq \tau(s,a) \geq \lambda_1(s,a).$$

The objective values are then identical by a simple algebraic manipulation.

Second, we need to show that for any solution of (ABP-MILP), there is a solution of (ABP–L_∞) with the same objective value. This follows by simple algebraic manipulation. □

C.9 Solving Small-Dimensional Linear Programs

Proposition 8.10. *In the proposed triangulation, the sub-polyhedra do not overlap and they cover the whole feasible set Y, given that the pivot point is in the interior of S.*

Proof. We prove the theorem by induction on the number of polyhedron splits that were performed. The base case is trivial: there is only a single polyhedron, which covers the whole feasible region.

For the inductive case, we show that for any polyhedron S the sub-polyhedra induced by the pivot point $\hat{y}$ cover S and do not overlap. The notation we use is the following: T denotes the original polyhedron and $\hat{y} = Tc$ is the pivot point, where $\mathbf{1}^{\mathsf{T}} c = 1$ and $c \geq \mathbf{0}$. Note that T is a matrix and $c, d, \hat{y}$ are vectors, and β is a scalar.

We show that the sub-polyhedra cover the original polyhedron S as follows. Take any $a = Td$ such that $\mathbf{1}^{\mathsf{T}} d = 1$ and $d \geq \mathbf{0}$. We show that there exists a sub-polyhedron that contains a and has $\hat{y}$ as a vertex. First, let

$$\hat{T} = \begin{pmatrix} T \\ \mathbf{1}^{\mathsf{T}} \end{pmatrix}$$

This matrix is square and invertible, since the polyhedron is non-empty. To get a representation of a that contains $\hat{y}$, we show that there is a vector o such that for some i, $o(i) = 0$:

$$\begin{aligned} \begin{pmatrix} a \\ 1 \end{pmatrix} = \hat{T}d &= \hat{T}o + (\beta\hat{y}) \\ o &\geq \mathbf{0}, \end{aligned}$$

for some $\beta > 0$. This will ensure that a is in the sub-polyhedron with $\hat{y}$ with vertex i replaced by $\hat{y}$. The value o depends on β as follows:

$$o = d - \beta\hat{T}^{-1} \begin{pmatrix} \hat{y} \\ 1 \end{pmatrix}.$$

This can be achieved by setting:

$$\beta = \min_i \frac{d(i)}{(\hat{T}^{-1}\hat{y})(i)}.$$

Since both d and $c = \hat{T}^{-1}\hat{y}$ are non-negative. This leaves us with an equation for the sub-polyhedron containing the point a. Notice that the resulting polyhedron may be of a smaller dimension than n when $o(j) = 0$ for some $i \neq j$.

To show that the polyhedra do not overlap, assume there exists a point a that is common to the interior of at least two of the polyhedra. That is, assume that a is a convex combination of the vertices:

$$\begin{aligned} a &= T_3 c_1 + h_1\hat{y} + \beta_1 y_1 \\ a &= T_3 c_2 + h_2\hat{y} + \beta_2 y_2, \end{aligned}$$

where T_3 represents the set of points common to the two polyhedra, and y_1 and y_2 represent the disjoint points in the two polyhedra. The values h_1, h_2, β_1, and β_2 are all scalars, while

c_1 and c_2 are vectors. Notice that the sub-polyhedra differ by at most one vertex. The coefficients satisfy:

$$\begin{aligned} c_1 &\geq \mathbf{0} & c_2 &\geq \mathbf{0} \\ h_1 &\geq 0 & h_2 &\geq 0 \\ \beta_1 &\geq 0 & \beta_2 &\geq 0 \\ \mathbf{1}^\mathsf{T} c_1 + h_1 + \beta_1 &= 1 & \mathbf{1}^\mathsf{T} c_2 + h_2 + \beta_2 &= 1 \end{aligned}$$

Since the interior of the polyhedron is non-empty, this convex combination is unique.

First assume that $h = h_1 = h_2$. Then we can show the following:

$$\begin{aligned} a = T_3 c_1 + h\hat{y} + \beta_1 y_1 &= T_3 c_2 + h\hat{y} + \beta_2 y_2 \\ T_3 c_1 + \beta_1 y_1 &= T_3 c_2 + \beta_2 y_2 \\ \beta_1 y_1 &= \beta_2 y_2 \\ \beta_1 = \beta_2 &= 0 \end{aligned}$$

This holds since y_1 and y_2 are independent of T_3 when the polyhedron is nonempty and $y_1 \neq y_2$. The last equality follows from the fact that y_1 and y_2 are linearly independent. This is a contradiction, since $\beta_1 = \beta_2 = 0$ implies that the point a is not in the interior of two polyhedra, but at their intersection.

Finally, assume WLOG that $h_1 > h_2$. Now let $\hat{y} = T_3\hat{c} + \alpha_1 y_1 + \alpha_2 y_2$, for some scalars $\alpha_1 \geq 0$ and $\alpha_2 \geq 0$ that represent a convex combination. We get:

$$\begin{aligned} a = T_3 c_1 + h_1\hat{y} + \beta_1 y_1 = T_3(c_1 + h_1\hat{c}) + (h_1\alpha_1 + \beta_1)y_1 + h_1\alpha_2 y_2 \\ a = T_3 c_2 + h_2\hat{y} + \beta_2 y_2 = T_3(c_2 + h_2\hat{c}) + h_2\alpha_1 y_1 + (h_2\alpha_2 + \beta_2)y_2. \end{aligned}$$

The coefficients sum to one as shown below.

$$\begin{aligned} \mathbf{1}^\mathsf{T}(c_1 + h_1\hat{c}) + (h_1\alpha_1 + \beta_1) + h_1\alpha_2 = \mathbf{1}^\mathsf{T} c_1 + \beta_1 + h_1(\mathbf{1}^\mathsf{T}\hat{c} + \alpha_1 + \alpha_2) = \mathbf{1}^\mathsf{T} c_1 + \beta_1 + h_1 = 1 \\ \mathbf{1}^\mathsf{T}(c_2 + h_2\hat{c}) + \alpha_1 + (h_2\alpha_2 + \beta_2) = \mathbf{1}^\mathsf{T} c_2 + \beta_2 + h_2(\mathbf{1}^\mathsf{T}\hat{c} + \alpha_1 + \alpha_2) = \mathbf{1}^\mathsf{T} c_2 + \beta_2 + h_2 = 1 \end{aligned}$$

Now, the convex combination is unique, and therefore the coefficients associated with each vertex for the two representations of a must be identical. In particular, equating the coefficients for y_1 and y_2 results in the following:

$$\begin{aligned} h_1\alpha_1 + \beta_1 &= h_2\alpha_1 & h_1\alpha_2 &= h_2\alpha_2 + \beta_2 \\ \beta_1 &= h_2\alpha_1 - h_1\alpha_1 & \beta_2 &= h_1\alpha_2 - h_2\alpha_2 \\ \beta_1 &= \alpha_1(h_2 - h_1) > 0 & \beta_2 &= \alpha_2(h_1 - h_2) < 0 \end{aligned}$$

We have that $\alpha_1 > 0$ and $\alpha_2 > 0$ from the fact that $\hat{y}$ is in the interior of the polyhedron S. Then, having $\beta_2 \leq 0$ is a contradiction with a being a convex combination of the vertices of S. □

Theorem 8.4. *Let f^* and $\tilde{f}^*$ be optimal solutions of (8.1) and (8.5) respectively. Then:*

$$\epsilon = |f^* - \tilde{f}^*| \leq \sqrt{\lambda}.$$

Moreover, this is the maximal linear dimensionality reduction possible with this error without considering the constraint structure.

Proof. We first show that indeed the error is at most $\sqrt{\lambda}$ and that any linearly compressed problem with the given error has at least f dimensions. Using a mapping that preserves the feasibility of both programs, the error is bounded by:

$$\epsilon \leq \left| f\left(w, x, \begin{pmatrix} F & G \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, z\right) - \tilde{f}\left(w, x, y_1, \begin{pmatrix} y_2 \\ z \end{pmatrix}\right)\right| = \left| x^\mathsf{T} C G y_2 \right|.$$

Denote the feasible region of y_2 as Y_2. From the orthogonality of $[F, G]$, we have that $\|y_2\|_2 \leq 1$ as follows:

$$\begin{aligned} y &= \begin{pmatrix} F & G \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ \begin{pmatrix} F^\mathsf{T} \\ G^\mathsf{T} \end{pmatrix} y &= \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \\ G^\mathsf{T} y &= y_2 \\ \|G^\mathsf{T} y\|_2 &= \|y_2\|_2 \end{aligned}$$

Then we have:

$$\begin{aligned} \epsilon &\leq \max_{y_2 \in Y_2} \max_{x \in X} \left| x^\mathsf{T} C G y_2 \right| \leq \max_{y_2 \in Y_2} \|CGy_2\|_2 \\ &\leq \max_{y_2 \in Y_2} \sqrt{y_2^\mathsf{T} G^\mathsf{T} C^\mathsf{T} C G y_2} \leq \max_{y_2 \in Y_2} \sqrt{y_2^\mathsf{T} L y_2} \leq \sqrt{\lambda} \end{aligned}$$

The result follows from Cauchy-Schwartz inequality, the fact that $C^\mathsf{T} C$ is symmetric, and Assumption 8.3. The matrix L denotes a diagonal matrix of eigenvalues corresponding to eigenvectors of G.

Now, let H be an arbitrary matrix that satisfies the preceding error inequality for G. Clearly, $H \cap F = \emptyset$, otherwise $\exists y$, $\|y\|_2 = 1$, such that $\|CHy\|_2 > \epsilon$. Therefore, we have $|H| \leq n - |F| \leq |G|$, because $|H| + |F| = |Y|$. Here $|\cdot|$ denotes the number of columns of the matrix. □

C.9.1 Sum of Convex and Concave Functions

In this section we show that the best-response function $g(y)$ may not be convex when the program is not in a semi-compact form. The convexity of the best-response function is crucial in bounding the approximation error and in eliminating the dominated regions.

We show that when the program is not in a semi-compact form, the best-response function can we written as a sum of a convex function and a concave function. To show that consider the following bilinear program.

$$\begin{aligned} \max_{w,x,y,z} \quad & f = r_1^\mathsf{T} x + s_1^\mathsf{T} w + x^\mathsf{T} C y + r_2^\mathsf{T} y + s_2^\mathsf{T} z \\ \text{s.t.} \quad & A_1 x + B_1 w = b_1 \\ & A_2 y + B_2 z = b_2 \\ & w, x, y, z \geq \mathbf{0} \end{aligned} \tag{C.18}$$

This problem may be reformulated as:

$$\begin{aligned} f &= \max_{\{y,z \,|\, (y,z)\in Y\}} \max_{\{x,w \,|\, (x,w)\in X\}} r_1^{\mathsf{T}} x + s_1^{\mathsf{T}} w + x^{\mathsf{T}} C y + r_2^{\mathsf{T}} y + s_2^{\mathsf{T}} z \\ &= \max_{\{y,z \,|\, (y,z)\in Y\}} g'(y) + s_2^{\mathsf{T}} z, \end{aligned}$$

where

$$g'(y) = \max_{\{x,w \,|\, (x,w)\in X\}} r_1^{\mathsf{T}} x + s_1^{\mathsf{T}} w + x^{\mathsf{T}} C y + r_2^{\mathsf{T}} y.$$

Notice that function $g'(y)$ is convex, because it is a maximum of a set of linear functions. Since $f = \max_{\{y \,|\, (y,z)\in Y\}} g(y)$, the best-response function $g(y)$ can be expressed as:

$$\begin{aligned} g(y) &= \max_{\{z \,|\, (y,z)\in Y\}} g'(y) + s_2^{\mathsf{T}} z = g'(y) + \max_{\{z \,|\, (y,z)\in Y\}} s_2^{\mathsf{T}} z \\ &= g'(y) + t(y), \end{aligned}$$

where

$$t(y) = \max_{\{z \,|\, A_2 y + B_2 z = b_2,\ y,z \geq \mathbf{0}\}} s_2^{\mathsf{T}} z.$$

Function $g'(y)$ does not depend on z, and therefore could be taken out of the maximization. The function $t(y)$ corresponds to a linear program, and its dual using the variable q is:

$$\begin{aligned} \min_{q} \quad & (b_2 - A_2 y)^{\mathsf{T}} q \\ \text{s.t.} \quad & B_2^{\mathsf{T}} q \geq s_2 \end{aligned} \tag{C.19}$$

Therefore:

$$t(y) = \min_{\{q \,|\, B_2^{\mathsf{T}} q \geq s_2\}} (b_2 - A_2 y)^{\mathsf{T}} q,$$

which is a concave function, because it is a minimum of a set of linear functions. The best-response function can now be written as:

$$g(y) = g'(y) + t(y),$$

which is a sum of a convex function and a concave function, also known as a d.c. function (Horst & Tuy, 1996). Using this property, it is easy to construct a program such that $g(y)$ will be convex on one part of Y and concave on another part of Y, as the following example shows. Note that in semi-compact bilinear programs $t(y) = 0$, which guarantees the convexity of $g(y)$.

Example C.25. *Consider the following bilinear program:*

$$\begin{aligned} \max_{x,y,z} \quad & -x + xy - 2z \\ s.t. \quad & -1 \leq x \leq 1 \\ & y - z \leq 2 \\ & z \geq 0 \end{aligned} \tag{C.20}$$

A plot of the best response function for this program is shown in Figure C.3.

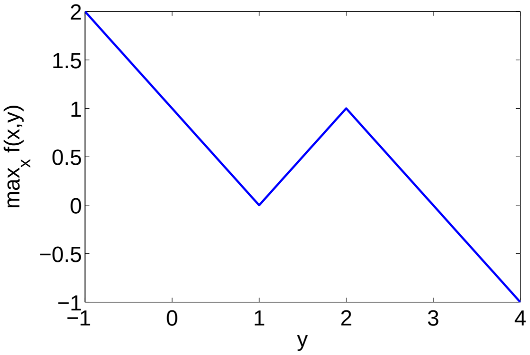

Figure C.3. A plot of a non-convex best-response function g for a bilinear program, which is not in a semi-compact form.

C.10 Sampling Bounds

Theorem 9.4. *Assume Assumption 9.1 and that there is a sample mapping function $\chi : \mathcal{S} \to \bar{\Sigma}$ such that for all $v \in \mathcal{M}(\psi)$:*

$$\max_{s \in \mathcal{S}} |(v - Lv)(s) - (v - \bar{L}v)(\chi(s))| \le \epsilon(\psi).$$

Then, the state selection error in Assumption 2.26 can be bounded as: $\epsilon_p(\psi) \le \epsilon(\psi)$.

Proof. Assume that $v \in \mathcal{M}(\psi) \cap \bar{\mathcal{K}}$. Then, using the fact that the Bellman residual is nonnegative for the sampled states, we have that:

$$\begin{aligned} -\epsilon_p = \min_{s \in \mathcal{S}} (v - Lv)(s) &\ge \min_{s \in \mathcal{S}} (v - Lv)(s) - \min_{s' \in \bar{\Sigma}} (v - \bar{L}v)(s') \\ &\ge \min_{s \in \mathcal{S}} \max_{s' \in \bar{\Sigma}} (v - Lv)(s) - (v - \bar{L}v)(s') \\ &\ge \min_{s \in \mathcal{S}} (v - Lv)(s) - (v - \bar{L}v)(\chi(s)) \\ &\ge -\max_{s \in \mathcal{S}} \left|(v - Lv)(s) - (v - \bar{L}v)(\chi(s))\right| \\ &\ge -\epsilon_p(\psi) \end{aligned}$$

□

Proposition 9.7. *Assume Assumption 9.1, Assumption 9.5, and a state mapping function*

$$\chi(s) = \arg\min_{\bar{s} \in \bar{\Sigma}} \|k(s) - k(\bar{s})\|.$$

such that $\max_{s \in \mathcal{S}} \|k(s) - k(\chi(s))\| \le q$ for some $q \in \mathbb{R}$. Then, Assumption 2.26 holds with the following constant:

$$\epsilon_p(\psi) = qK_r + q\psi(K_\phi + \gamma K_p).$$

Proof. First, we can set the following values in Assumption 9.6.

$$\sigma_\phi = qK_\phi \qquad \sigma_r = qK_r \qquad \sigma_p = qK_p$$

The following shows the derivation for σ_p; the other derivations are very similar.

$$\sigma_p \le \|\phi(s) - \phi(\chi(s))\|_{\infty, e_{-1}} \le K_\phi \|k(\chi(s)) - k(s)\| \le qK_\phi.$$

Now, we show that the hypothesis of Theorem 9.4 holds, as follows.

$$
\begin{aligned}
\max_{s\in\mathcal{S}} &|(v - Lv)(s) - (v - \bar{L}v)(\chi(s))| \\
&\le \max_{s\in\mathcal{S},a\in\mathcal{A}} |(v - \gamma P_a v - r_a)(s) - (v - \gamma P_a v - r_a)(\chi(s))| \\
&\le \max_{s\in\mathcal{S},a\in\mathcal{A}} |\mathbf{1}_s^\mathsf{T}(\Phi x - \gamma P_a \Phi x - r_a) - \mathbf{1}_{\chi(s)}^\mathsf{T}(\Phi x - \gamma P_a \Phi x - r_a)| \\
&\le \max_{s\in\mathcal{S},a\in\mathcal{A}} |(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T})\Phi x| + |(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T})\gamma P_a \Phi x| + \\
&\qquad + |(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T}) r_a| \\
&\le \max_{s\in\mathcal{S},a\in\mathcal{A}} \|(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T})\Phi\|_{\infty,e_{-1}} \|x\|_{1,e} + \\
&\qquad + \|(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T})\gamma P_a \Phi\|_{\infty,e_{-1}} \|x\|_{1,e} + \|(\mathbf{1}_s^\mathsf{T} - \mathbf{1}_{\chi(s)}^\mathsf{T}) r_a\|_{\infty,e_{-1}} \\
&\le q K_r + q\psi(K_\phi + \gamma K_p)
\end{aligned}
$$

We used the fact that L and $\bar{L}$ are identical for states that are sampled and Lemma 9.2. □

Theorem 9.8. *Assume Assumption 9.1 and a sample mapping function $\chi : (\mathcal{S} \times \mathcal{A}) \to \bar{\Sigma}^{n+1}$, where n is the dimensionality of the state–action embedding function $k(\mathcal{S}, \mathcal{A})$ and $\bar{\Sigma}^{n+1}$ denotes subsets of length $n + 1$. The sample mapping function χ satisfies for all $s, a \in \mathcal{S}, \mathcal{A}$:*

$$
\begin{aligned}
k(s,a) &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) k(s_i,a_i) \\
1 &= \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a)
\end{aligned}
$$

for some function $\beta_i(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ and in addition for some $\sigma_\phi, \sigma_r \in \mathbb{R}$:

$$
\left\| \left(\phi(s) - \gamma P(s,a)^\mathsf{T}\Phi\right) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a)\left(\phi(s_i) - \gamma P(s_i,a_i)^\mathsf{T}\Phi\right) \right\|_{\infty,e_{-1}} \le \sigma_\phi
$$

$$
\left| r(s,a) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) r(s_i,a_i) \right| \le \sigma_r.
$$

Then, the following holds:

$$
\epsilon_p(\psi) \le \sigma_r + \sigma_\phi \psi.
$$

Proof. Assume $v \in \bar{\mathcal{K}} \cap \mathcal{M}(\psi)$. The let $\bar{s}, \bar{a}$ be a minimizer of:

$$
-\epsilon_p = \min_{s\in\mathcal{S},a\in\mathcal{A}} (v - L_a v)(s) = \min_{s\in\mathcal{S},a\in\mathcal{A}} v(s) - \gamma P(s,a)^\mathsf{T} v - r(s,a).
$$

To simplify the notation, define:

$$
q(s,a) = \phi(s) - \gamma P(s,a)^\mathsf{T}\Phi.
$$

Then, using that for $s, a \in \bar{\Sigma}$, $q(s,a)^\mathsf{T} x - r(s,a) \geq 0$ and $\beta_i \geq 0$, the theorem is derived as follows:

$$\begin{aligned}
-\epsilon_p &= v(\bar{s}) - \gamma P(\bar{s},\bar{a})^\mathsf{T} v - r(\bar{s},\bar{a}) = \phi(\bar{s})^\mathsf{T} x - \gamma P(\bar{s},\bar{a})^\mathsf{T} \Phi x - r(\bar{s},\bar{a}) \\
&\geq \left(q(\bar{s},\bar{a})^\mathsf{T} x - r(\bar{s},\bar{a})\right) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) \left(q(s_i,a_i)^\mathsf{T} x - r(s_i,a_i)\right) \\
&\geq - \left| q(\bar{s},\bar{a})^\mathsf{T} x - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) q(s_i,a_i)^\mathsf{T} x \right| - \left| r(\bar{s},\bar{a}) - \sum_{i=1}^{n+1} \beta_i(s,a) - r(s_i,a_i) \right| \\
&\geq - \left\| q(\bar{s},\bar{a}) - \sum_{(s_i,a_i)\in\chi(s,a)} \beta_i(s,a) q(s_i,a_i) \right\|_{\infty,e_{-1}} \|x\|_{1,e} - \left| r(\bar{s},\bar{a}) - \sum_{i=1}^{n+1} \beta_i(s,a) r(s_i,a_i) \right| \\
&\geq -\sigma_r - \sigma_\phi \psi
\end{aligned}$$

Here, we used Lemma 9.2 and the fact that $(q(s,a))(1) = (1-\gamma)$ from Assumption 9.1. □

Theorem 9.13. *Assume Assumption 9.1, Assumption 9.6, and Assumption 9.12 and let $v \in \mathcal{M}(\psi)$ be a representable value function. Then, Assumption 2.27 is satisfied with the following constant:*

$$\epsilon_c = \sigma_s \|\Phi^\mathsf{T} \bar{c}\|_{\infty,e_{-1}} \psi + 2\sigma_\phi \psi$$

when $c = \alpha$ and $e = (0 \quad \mathbf{1}^\mathsf{T})^\mathsf{T}$.

Proof. Recall that:

$$\bar{c}^\mathsf{T} \Phi = \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s} \sum_{s\in\bar{\Sigma}} c(s)\phi(s) \qquad c^\mathsf{T} \Phi = \sum_{s\in\mathcal{S}} c(s)\phi(s)$$

The main idea of the proof is to individually bound the error due to the difference between s and $\chi(s)$ and also the uniformity of sampling.

$$\begin{aligned}
\left\|\Phi^{\mathsf{T}} c - \Phi^{\mathsf{T}} \bar{c}\right\|_{\infty,e_{-1}} &= \left\|\sum_{s\in\mathcal{S}} c(s)\phi(s) - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s)\right\|_{\infty,e_{-1}} \\
&= \left\|\sum_{s\in\mathcal{S}} c(s)\phi(s) - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \right. \\
&\quad \left. + \left(\sum_{s\in\mathcal{S}} c(s)\phi(s) - c(\chi(s))\phi(\chi(s))\right) - \left(\sum_{s\in\mathcal{S}} c(s)\phi(s) - c(\chi(s))\phi(\chi(s))\right)\right\|_{\infty,e_{-1}} \\
&= \left\| - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \right. \\
&\quad \left. + \left(\sum_{s\in\mathcal{S}} c(s)\phi(s) - c(\chi(s))\phi(\chi(s))\right) + \left(\sum_{s\in\mathcal{S}} c(\chi(s))\phi(\chi(s))\right)\right\|_{\infty,e_{-1}} \\
&\le \left\| - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \sum_{s\in\mathcal{S}} c(\chi(s))\phi(\chi(s))\right\|_{\infty,e_{-1}} + \\
&\quad \left\|\sum_{s\in\mathcal{S}} \left(c(s)\phi(s) - c(\chi(s))\phi(\chi(s))\right)\right\|_{\infty,e_{-1}}
\end{aligned}$$

Now,

$$\left\|\sum_{s\in\mathcal{S}} \left(c(s)\phi(s) - c(\chi(s))\phi(\chi(s))\right)\right\|_{\infty,e_{-1}} \le \left|\sum_{s\in\mathcal{S}} (c(s) - c(\chi(s)))\right| \sigma_\phi \le 2\sigma_\phi,$$

using the fact that $e = (0 \quad \mathbf{1}^{\mathsf{T}})^{\mathsf{T}}$ and $c = \alpha$. And finally:

$$\begin{aligned}
&\left\| - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\ \sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \sum_{s\in\mathcal{S}} c(\chi(s))\phi(\chi(s))\right\|_{\infty,e_{-1}} \\
&\qquad = \left\| - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \sum_{\bar{s}\in\bar{\Sigma}}\ \sum_{s\in\chi^{-1}(\bar{s})} c(\chi(s))\phi(\chi(s))\right\|_{\infty,e_{-1}} \\
&\qquad = \left\| - \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s}\sum_{s\in\bar{\Sigma}} c(s)\phi(s) + \sum_{\bar{s}\in\bar{\Sigma}} \left|\chi^{-1}(\bar{s})\right| c(\bar{s})\phi(\bar{s})\right\|_{\infty,e_{-1}} \\
&\qquad = \left|\frac{\left|\chi^{-1}(\bar{s})\right| |\bar{\Sigma}|_s}{|\mathcal{S}| - 1}\right| \left\|\sum_{\bar{s}\in\bar{\Sigma}} \frac{|\mathcal{S}|}{|\bar{\Sigma}|_s} c(\bar{s})\phi(\bar{s})\right\|_{\infty,e_{-1}} \\
&\qquad \le \sigma_s \|\bar{c}^{\mathsf{T}}\Phi\|_{\infty,e_{-1}}
\end{aligned}$$

The theorem then follows by putting the inequalities together and using Lemma 9.2 to bound $|(c - \bar{c})\Phi v|$. □

Theorem 9.14. *Assume that samples $\tilde{\Sigma}$ are available and that the number of samples per each state and action pair is at least n. Also assume that $e = (0\mathbf{1}^\mathsf{T})^\mathsf{T}$. Then, the transition estimation error ϵ_s (Assumption 2.28) is bounded as:*

$$\begin{aligned}\mathbf{P}\left[\epsilon_s(\psi) > \epsilon\right] &\le Q\left(2|\phi| \exp\left(\frac{2(\epsilon/(\psi\cdot\gamma))^2}{M_\phi n}\right), |\tilde{\Sigma}|_a\right) \\ &\le 2|\tilde{\Sigma}|_a |\phi| \exp\left(\frac{2(\epsilon/(\psi\cdot\gamma))^2}{M_\phi n}\right),\end{aligned}$$

where $|\tilde{\Sigma}|_a$ is the number of sampled state-action pairs,

$$M_\phi = \max_{s\in\mathcal{S}} \|\phi(s)\|_\infty.$$

and

$$Q(x, y) = 1 - (1 - x)^y.$$

Proof. The probability for the transition estimation error that we need to bound is defined as follows, where n is the number of samples per each state–action pair in $\tilde{\Sigma}$.

$$\begin{aligned}\mathbf{P}\left[\epsilon_s(\psi) > \epsilon\right] &\le \mathbf{P}\left[\max_{v\in\mathcal{M}} \max_{s\in\tilde{\Sigma}} |(\bar{L}v)(s) - (\tilde{L}v)(s)| > \epsilon\right] \\ &\le \mathbf{P}\left[\max_{v\in\mathcal{M}} \max_{s,a\in\tilde{\mathcal{S}}} |(\bar{L}_a v)(s) - (\tilde{L}_s v)(s)| > \epsilon\right] \\ &= \mathbf{P}\left[\max_{v\in\mathcal{M}} \max_{s,a\in\tilde{\mathcal{S}}} |\gamma P(s,a)^\mathsf{T} v + r(s,a) - \gamma\frac{1}{n}\sum_{f=1}^{n} P(s,a,s_j)^\mathsf{T} v - r(s,a)| > \epsilon\right] \\ &= \mathbf{P}\left[\max_{v\in\mathcal{M}} \max_{s,a\in\tilde{\mathcal{S}}} |P(s,a)^\mathsf{T} v - \frac{1}{n}\sum_{j=1}^{n} P(s,a,s_j)^\mathsf{T} v| > \epsilon/\gamma\right] \\ &= \mathbf{P}\left[\max_{v\in\mathcal{M}} \max_{s,a\in\tilde{\mathcal{S}}} |P(s,a)^\mathsf{T} v - \frac{1}{n}\sum_{j=1}^{n} P(s,a,s_j)^\mathsf{T} v| > \epsilon/\gamma\right] \\ &= \mathbf{P}\left[\max_{s,a\in\tilde{\mathcal{S}}} \max_{f=1\ldots|\phi|} \psi|P(s,a)^\mathsf{T} \phi_f - \frac{1}{n}\sum_{j=1}^{n} P(s,a,s_j)^\mathsf{T} \phi_f| > \epsilon/\gamma\right] \\ &\le |\phi| \max_{f=1\ldots|\phi|} \mathbf{P}\left[\max_{s,a\in\tilde{\mathcal{S}}} |P(s,a)^\mathsf{T} \phi_f - \frac{1}{n}\sum_{j=1}^{n} P(s,a,s_j)^\mathsf{T} \phi_f| > \epsilon/(\gamma\cdot\psi)\right]\end{aligned}$$

Here, we used the union bound for features. Note that the error for $\phi_1 = \mathbf{0}$ is 0. Now, let

$$X_i = |P(s_i,a_i)^\mathsf{T} \phi_f - \frac{1}{n}\sum_{j=1}^{n} P(s_i,a_i,s_{ij})^\mathsf{T} \phi_f|,$$

for *some* feature f and $i \in \tilde{\Sigma}$ for simplicity. We then have using the fact that the random variables X_i are independent:

$$\begin{aligned}\mathbf{P}\left[\max_{s,a\in\tilde{\mathcal{S}}} |P(s,a)^\mathsf{T}\phi_f - \frac{1}{n}\sum_{j=1}^{n} P(s,a,s_{ij})^\mathsf{T}\phi_f| > \epsilon/(\gamma\cdot\psi)\right] &= \mathbf{P}\left[\max_{i\in\tilde{\mathcal{S}}} X_i > \epsilon/(\gamma\cdot\psi)\right] \\ &= Q(\mathbf{P}\left[X_i > \epsilon/(\gamma\cdot\psi)\right], |\Sigma|_a).\end{aligned}$$

The theorem then follows from Hoeffding's inequality applied to $\mathbf{P}\left[X_i > \epsilon/(\gamma\cdot\psi)\right]$ and the linear upper bound on the function $Q(x,\cdot)$. □

Proposition 9.15. *The bound in Theorem 9.14 is asymptotically tight with respect to* $|\tilde{\Sigma}|_a$.

Proof. To prove the tightness of the bound, consider the following MDP. The states are $\mathcal{S} = \{\bar{s}_0, \bar{s}_1, s_1, s_2, \ldots\}$ and there is a single action $\mathcal{A} = \{a_1\}$. The transitions are defined as follows:

$$\begin{aligned} P(\bar{s}_0, a_1, \bar{s}_0) &= 1 & \qquad P(\bar{s}_1, a_1, \bar{s}_1) &= 1 \\ P(s_i, a_1, \bar{s}_0) &= 0.5 & \qquad P(s_i, a_1, \bar{s}_1) &= 0.5 \end{aligned}$$

The rewards are all zero. Consider a feature defined as follows:

$$\phi_2(\bar{s}_0) = 0 \quad \phi_2(\bar{s}_1) = 1 \quad \phi_2(s_i) = 0,$$

with the $\phi_1 = \mathbf{1}$. The theorem then follows by simply computing the probability of the violation of the error as a function of the number of states s_i included in the sample. The error can then be bounded as:

$$\begin{aligned}\mathbf{P}\left[\max_{i\in\tilde{\Sigma}} \frac{1}{m}\sum_{j=1}^{m} X_i(\omega_j) - \mathbf{E}\left[X_i\right]\right] &= \mathbf{P}\left[\max_{i\in\tilde{\Sigma}} \frac{1}{m}\sum_{j=1}^{m} X_i(\omega_j)\right] \\ &\le \mathbf{P}\left[\sum_{j=1}^{m} \max_{i\in\tilde{\Sigma}} \frac{1}{m} X_i(\omega_j)\right] \\ &\le \mathbf{P}\left[\sum_{j=1}^{m} \max_{i\in\tau(m)} \frac{1}{m} X_i(\omega_j)\right] \\ &\le \mathbf{P}\left[\sum_{j=1}^{m} \sum_{i\in\tau(m)} \frac{1}{m} X_i(\omega_j)\right] \\ &\le |\tau(m)|\mathbf{P}\left[\frac{1}{m}\sum_{j=1}^{m} X_i(\omega_j) > \frac{\epsilon}{|\tau(m)|}\right]\end{aligned}$$

□

Theorem 9.17. *Assume that samples $\tilde{\Sigma}$ are generated using* common random numbers *and that the number samples of common numbers is m. Then:*

$$\{\mathrm{I}\left\{z(s,a,\omega)v \geq \epsilon\right\} \mid \omega \in \Omega, v \in \mathcal{M}(\psi), \epsilon \in \mathbb{R}_+\}.$$

Then, the transition estimation error ϵ_s (Assumption 2.28) is bounded as:

$$\mathbf{P}\left[\epsilon_s(\psi) > \epsilon\right] \leq 2|\phi| \exp\left(\frac{2(\epsilon/(\psi \cdot \gamma \cdot |\tau(m)|))^2}{M_\phi m}\right),$$

where $|\tilde{\Sigma}|_a$ is the number of sampled state-action pairs and

$$M_\phi = \max_{s \in \mathcal{S}} \|\phi(s)\|_\infty.$$

Proof. The first part of the proof follows similarly to the proof of Theorem 9.14. To simplify the notation, we use

$$X_i(\omega_j) = |P(s_i,a_i)^\mathsf{T}\phi_f - \frac{1}{n}\sum_{j=1}^{n} P(s_i,a_i,z(s_i,a_i,\omega_j))^\mathsf{T}\phi_f|$$

for *some* feature f and $i \in \tilde{\Sigma}$. These random variables are not necessarily independent, as they are in Theorem 9.14. Now, we have from $\mathbf{E}\left[X_i\right] = 0$ that:

$$\mathbf{P}\left[\max_{i \in \tilde{\Sigma}} \frac{1}{m}\sum_{j=1}^{m} X_i()\right]$$

□

Proposition 9.19. *Assume a triangulation $T = \{T_1 \ldots T_m\}$ of $k(\mathcal{S})$ where n is the dimension of $k(\mathcal{S})$. In addition, assume that $\tilde{\rho}_1$ and $\tilde{\rho}_2$ are Gaussian processes with* a posteriori $1-\delta$ *confidence lower bounds l_1, l_2 and upper bounds u_1, u_2. Note that the confidence bounds on a set of values is not the same as the union of individual confidence bounds. Then, the hypothesis of Theorem 9.8 holds with the minimal possible value of the following constants:*

$$\sigma_\phi \geq \max_{T_j \in T} \max_{i \in |\phi|} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_1(T_j(l), i) - l_1\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right)$$

$$\sigma_\phi \geq \max_{T_j \in T} \max_{i \in |\phi|} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} u_1\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right) - \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_1(T_j(l), i)$$

$$\sigma_r \geq \max_{T_j \in T} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_2(T_j(l)) - l_2\left(\sum_{l=1}^{n+1} \beta_l T_j(l), i\right)$$

$$\sigma_r \geq \max_{T_j \in T} \max_{\beta_1 \ldots \beta_{n+1} \in \ B} u_2\left(\sum_{l=1}^{n+1} \beta_l T_j(l)\right) - \sum_{l=1}^{n+1} \beta_l \tilde{\rho}_2(T_j(l))$$

where $B = \{\beta_1 \ldots \beta_{n+1} \,\big|\, \sum_{l=1}^{n+1} \beta_l = 1, \beta_l \geq 0\}$.

Proof. To prove the theorem, we have to define the sample selection function χ_g to be used with Theorem 9.8. The corollary simply follows by setting χ_g to map a states to the polyhedron of the samples that it is included in. □

C.11 Feature Selection

Proposition 10.4. *The function $\theta_L(\psi)$ is convex and piecewise linear.*

Proof. Assume without loss of generality that $x \geq \mathbf{0}$. Then (ALP–R) becomes:

$$\begin{aligned}\min_{x} \quad & c^{\mathsf{T}}\Phi \\ \text{s.t.} \quad & A\Phi x \geq b \\ & e^{\mathsf{T}} x \leq \psi \\ & x \geq \mathbf{0}\end{aligned}$$

The proof is straight-forward from the dual formulation of (ALP–R). The dual formulation is:

$$\begin{aligned}\max_{y} \quad & b^{\mathsf{T}} y - \psi\lambda \\ \text{s.t.} \quad & \Phi^{\mathsf{T}} A^{\mathsf{T}} y + e\lambda \leq \Phi^{\mathsf{T}} c \\ & y \geq \mathbf{0} \\ & \lambda \geq 0\end{aligned}$$

Because, ψ does not influence the feasibility of a solution. Therefore, the function θ_L represents a maximum of a *finite* set of linear functions (corresponding to the basic feasible solutions) and is therefore convex and piecewise linear. □

C.12 Heuristic Functions

Proposition 11.3. *Given a complete set of samples, the heuristic function $v \in \mathcal{K}_2$ is admissible. That is, for all $s \in \mathcal{S}$, $v(s) \geq v^*(s)$.*

Proof. Let $v^* = (\mathbf{I} - \gamma P^*)^{-1} r^*$. Then, any feasible solution satisfies:

$$\begin{aligned}
\gamma P^* v + r^* &\leq v \\
0 &\leq (\mathbf{I} - \gamma P^*) v - r^* \\
0 &\leq v - (\mathbf{I} - \gamma P^*)^{-1} r^* \\
(\mathbf{I} - \gamma P^*)^{-1} r^* &\leq v \\
v^* &\leq v.
\end{aligned}$$

We used the monotonicity of $(\mathbf{I} - \gamma P^*)^{-1}$, which can be shown easily from its geometric series expansion. □

Lemma 11.7. *Assume that* $\mathbf{1}$ *is representable in* $\mathcal{M}$*. Then there exists a heuristic function* $\hat{v}$ *that is feasible in* (11.5) *and satisfies:*

$$\|\hat{v} - v^*\|_\infty \leq \frac{2}{1 - \gamma\alpha} \min_x \|v^* - \Phi x\|_\infty.$$

Proof. Let a closest representable heuristic function be $\tilde{v}$, defined as:

$$\epsilon = \|\tilde{v} - v^*\|_\infty = \min_x \|v^* - \Phi x\|_\infty.$$

This function may not satisfy the inequalities (11.5). We show that it is possible to construct $\hat{v}$ that satisfies the inequalities. From the assumption, we have that:

$$v^* - \epsilon \mathbf{1} \leq \tilde{v} \leq v^* + \epsilon \mathbf{1}.$$

and for $a \in \mathcal{A}$

$$\mathbf{0} \leq T_a \mathbf{1} \leq \gamma\alpha \mathbf{1},$$

directly from the definition of the inequalities (11.5). We use $\mathbf{0}$ to denote a zero vector of the appropriate size. Now let $\hat{v} = \tilde{v} + d\mathbf{1}$, for some d. Appropriate value of d to make $\hat{v}$ feasible can be derived using Lemma 11.5 as:

$$\begin{aligned}
\hat{v} &= \tilde{v} + d\mathbf{1} \\
&\geq v^* - \epsilon\mathbf{1} + d\mathbf{1} \\
&\geq T v^* + r + (d - \epsilon)\mathbf{1} \\
&\geq T(\tilde{v} - \epsilon\mathbf{1}) + r + (d - \epsilon)\mathbf{1} \\
&\geq T\tilde{v} - \epsilon\gamma\alpha\mathbf{1} + r + (d - \epsilon)\mathbf{1} \\
&= T\tilde{v} + r + (d - (\gamma\alpha + 1)\epsilon)\mathbf{1} \\
&= T(\hat{v} - d\mathbf{1}) + r + (d - (\gamma\alpha + 1)\epsilon)\mathbf{1} \\
&\geq T\hat{v} + r + ((1 - \gamma\alpha)d - (1 + \gamma\alpha)\epsilon)\mathbf{1}.
\end{aligned}$$

Therefore, $\hat{v} \geq T\hat{v} + r$ if:

$$\begin{aligned} ((1-\gamma\alpha)d - (1+\gamma\alpha)\epsilon)\mathbf{1} &\geq \mathbf{0} \\ d &\geq \frac{1+\gamma\alpha}{1-\gamma\alpha}\epsilon. \end{aligned}$$

Since $d \geq \epsilon$, also $\hat{v}(\tau) \geq 0$ is satisfied. Finally:

$$\|\hat{d} - v^*\|_\infty \leq \|\tilde{v} - v^*\|_\infty + \|d\mathbf{1}\|_\infty \leq \frac{2}{1-\gamma\alpha}\epsilon.$$

□

Lemma 11.11. *Assume that there exists a Lyapunov hierarchy $u^1 \ldots u^l$, such that each u^i is representable in $\mathcal{M}$. Then there exists a heuristic function $\hat{v}$ in Φ that is feasible in* (11.5)*, such that:*

$$\|\hat{v} - v^*\|_\infty \leq \left(\prod_{i=1}^{l} \frac{(1+\alpha\gamma)\max_{s\in\mathcal{S}} u^i(s)}{(1-\alpha\gamma\beta_i)\min_{s\in\mathcal{S}} u^i_i(s)}\right) 2\min_x \|v^* - \Phi x\|_\infty,$$

where u^i_i is the vector u^i restricted to states in partition i.

Proof. First, let

$$\epsilon = 2\|\tilde{v}_1 - v^*\|_\infty = 2\min_x \|\Phi x - v^*\|_\infty.$$

Construct $\tilde{v} = \tilde{v}_1 + \epsilon\mathbf{1}$ such that:

$$v^* \leq \tilde{v} \leq v^* + \epsilon.$$

The proof follows by induction on the size l of the Lyapunov hierarchy. Assume that the inequalities are satisfied for all $i' < i$, with the error ϵ and the property that the current $\tilde{v} \geq v^*$. Then let $\hat{v} = \tilde{v} + d\mathbf{1}$, for some d. Then, using Lemma 11.5, we have:

$$\begin{aligned} \hat{v} &= \tilde{v} + du^i \geq v^* + du^i \geq T_i v^* + r_d u^i \\ &\geq T_i\tilde{v} - \gamma\alpha\epsilon\mathbf{1} + r + du^i \\ &\geq T_i(\hat{v} - du^i) - \gamma\alpha\epsilon\mathbf{1} + r + du^i \\ &\geq T_i\hat{v} + r - \alpha\beta_i\gamma du^i + du^i - \gamma\alpha\epsilon e \end{aligned}$$

To satisfy $\hat{v} \geq T_i\hat{v} + r_i$, set d to:

$$\begin{aligned} \alpha\beta_i\gamma du^i + du^i &\geq \gamma\alpha\epsilon\mathbf{1} \\ d &\geq \frac{\gamma\alpha}{1-\alpha\beta_i\gamma}\frac{1}{\min_{s\in\mathcal{S}} u^i_i(s)}\epsilon. \end{aligned}$$

Therefore the total approximation error for $\hat{v}$ is:

$$\|\hat{v} - v^*\|_\infty \leq \frac{\gamma\alpha}{1-\alpha\beta_i\gamma}\frac{\max_{s\in\mathcal{S}} u^i(s)}{\min_{s\in\mathcal{S}} u^i_i(s)}\epsilon$$

The lemma follows because $d \geq 0$ and $u^i \geq 0$, and thus the condition $\tilde{v} \geq v^*$ is not violated. In the end, all the constraints are satisfied from the definition of the Lyapunov hierarchy. □

BIBLIOGRAPHY

Abbeel, P., Ganapathi, V., & Ng, A. Y. (2006). Learning vehicular dynamics, with application to modeling helicopters. In *Advances in Neural Information Processing Systems*, pp. 1–8.

Adelman, D. (2004). A price-directed approach to stochastic inventory/routing. *Operations Research*, *52*, 499–514.

Antos, A., Szepesvri, C., & Munos, R. (2008). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, *71*, 89–129.

Asif, S. (2008). Primal dual pursuit: A homotopy-method based algorithm for the Dantzig selector. Master's thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology.

Asif, S. (2009). Dantzig selector homotopy with dynamic measurements. In *IS&T/SPIE Computational Imaging VII.*

Asmuth, J., Li, L., Littman, M., Nouri, A., & Wingate, D. (2009). A bayesian sampling approach to exploration in reinforcement learning. In *Uncertainty in Artificial Intelligence.*

Auer, P., Jaksch, T., & Ortner, R. (2009). Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems.*

Axsater, S. (2006). *Inventory Control* (2nd edition). Springer.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pp. 30–37.

Barto, A., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1), 81–138.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2004). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Beliaeva, N., & Zilberstein, S. (2005). Generating admissible heuristics by abstraction for search in stochastic domains. In *Abstraction, Reformulation and Approximation*, pp. 14–29. Springer Berlin / Heidelberg.

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press.

Ben-Tal, A., & Nemirovski, A. (2008). Selected topics in robust optimization. *Mathematical Programming, Series B*, *112*, 125–158.

Bennett, K. P., & Mangasarian, O. L. (1992). Bilinear separation of two sets in n-space. Tech. rep., Computer Science Department, University of Wisconsin.

Benton, J., van den Briel, M., & Kambhampati, S. (2007). A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Bertsekas, D. P. (2003). *Nonlinear programming*. Athena Scientific.

Bertsekas, D. P., & Castalion, D. A. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transations on Automatic Control*, *34*, 589–598.

Bertsekas, D. P., & Ioffe, S. (1997). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Tech. rep. LIDS-P-2349, LIDS.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena Scientific.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5–33.

Bonet, B., & Geffner, H. (2003a). Faster heuristic search algorithms for planning under uncertainty and full feedback. In *International Joint Conference on Artificial Intelligence*.

Bonet, B., & Geffner, H. (2003b). Labeled rtdp: Improving the convergence of real-time dynamic programming. In *International Conference on Autonomous Planning (ICAPS)*.

Bonet, B., & Geffner, H. (2009). Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Bousquet, O., Boucheron, S., & Lugosi, G. (2004). Introduction to statistical learning theory. *Advanced Lectures on Machine Learning*, *3176*, 169–207.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In Mellish, C. (Ed.), *International Joint Conference on Artificial Intelligence*, pp. 1104–1111, San Francisco. Morgan Kaufmann.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Brunskill, E., Leffler, B. R., Li, L., Littman, M. L., & Roy, N. (2009). Provable efficient learning with typed parametric models. *Journal of Machine Learning Research*, *10*, 1955–1988.

Bylander, T. (1997). A linear programming heuristic for optimal planning. In *National Conference on Artificial Intelligence*, pp. 694–699.

Candes, E., & Tao, T. (2007). The Dantzig selector:statistical estimation when p is much larger than n. *Annals of Statistics*, *35*, 2313–2351.

Carpara, A., & Monaci, M. (2009). Bidimensional packing by bilinear programming. *Mathematical Programming Series A*, *118*, 75–108.

Cheng, H. T. (1988). *Algorithms for Partially Observable Markov Decision Processes.* Ph.D. thesis, University of British Columbia.

Culberson, J. C., & Schaeffer, J. (1994). Efficiently searching the 15-puzzle. Tech. rep., Department of Computer Science, University of Alberta.

Culberson, J. C., & Schaeffer, J. (1996). Searching with pattern databases. In *Advances in Artifical Intelligence*, pp. 402–416. Springer Berlin / Heidelberg.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3), 318–334.

de Farias, D. P. (2002). *The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application.* Ph.D. thesis, Stanford University.

de Farias, D. P., & van Roy, B. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, *29*(3), 462–478.

Devroye, L., Gyorfi, L., & Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition.* Springer.

Dinh, H., Russell, A., & Su, Y. (2007). On the value of good advice: The complexity of A* searchwith accurate heuristics. In *AAAI*.

Drager, K., Fingbeiner, B., & Podelski, A. (2006). Directed model checking with distance-preserving abstractions. In *International SPIN Workshop, LNCS*, Vol. 3925, pp. 19–34.

Dzeroski, S., de Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Edelkamp, S. (2001). Planning with pattern databases. In *ECP*.

Edelkamp, S. (2002). Symbolic pattern databases in heuristic search planning. In *AIPS*.

Edelkamp, S. (2006). Automated creation of pattern database search heuristics. In *Workshop on Model Checking and Artificial Intelligence*.

Edelkamp, S. (2007). Symbolic shortest paths planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Farahmand, A. M., Ghavamzadeh, M., Szepesvari, C., & Mannor, S. (2009). Regularized policy iteration. In Koller, D., Schuurmans, D., Bengio, Y., & Bottou, L. (Eds.), *Advances in Neural Information Processing Systems 21*, pp. 441–448.

Farias, V., & van Roy, B. (2006). *Probabilistic and Randomized Methods for Design Under Uncertainty*, chap. 6: Tetris: A Study of Randomized Constraint Sampling. Springer-Verlag.

Feng, Z., Hansen, E. A., & Zilberstein, S. (2003). Symbolic generalization for on-line planning. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 209–216.

Feng, Z., & Zilberstein, S. (2004). Region-based incremental pruning. In *Conference on Uncertainty in Artificial Intelligence*.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, *25*, 85–118.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(189-208), 189–208.

Forsund, F. R. (2009). *Hydropower Economics*. Springer US.

Friedlander, M. P., & Saunders, M. A. (2007). Discussion:the Dantzig selector:statistical estimation when p is much larger than n. *The Annals of Statistics*, *35*(6), 2385–2391.

Gaschnig, J. (1979). Ph.D. thesis, Carnegie-Mellon University.

Gerevini, A., & Long, D. (2005). Plan constraints and preferences in pddl3. Tech. rep., Dipartimento di Elettronica per l'Automazione, Universita degli Studi di Brescia.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Glasserman, P., & Yao, D. D. (1992). Some guidelines and guarantees for common random numbers. *Management Science*, *38*(6), 884–908.

Goldfarb, D., & Iyengar, G. (2003). Robust convex quadratically constrained programs. *Mathematical Programming*, *97*, 495–515.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. Tech. rep., Carnegie Mellon University. CMU-CS-95-103.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*, 399–468.

Gyorfi, L., Kohler, M., Lrzyzak, A., & Walk, H. (2002). *A Distribution-Free Theory of Nonparametric Regression*. Springer.

Hansen, E. A., & Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research*, *28*, 267–297.

Hansen, E. A., & Zilberstein, S. (2001). LAO *: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Haslum, P., Bonet, B., & Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *National Conference on AI*.

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In *National Conference on Artificial Intelligence*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd edition).

Helmert, M., & Mattmuller, R. (2008). Accuracy of admissible heuristic functions in selected planning domains. In *National Conference on Artificial Intelligence*.

Helmert, M., & Roger, G. (2008). How good is almost perfect. In *National Conference on AI.*

Hoey, J., & Poupart, P. (2005). Solving POMDPs with continuous or large discrete observation spaces. In *International Joint Conference on Artificial Intelligence.*

Holte, R. C., Grajkowski, J., & Tanner, B. (2005). Hierarchical heuristic search revisited. In *Abstraction, Reformulation and Approximation*, pp. 121–133. Springer Berlin / Heidelberg.

Holte, R. C., Mkadmi, T., Zimmer, R., & MacDonald, A. (1996a). Speeding up problem solving by abstraction: a graph oriented approach. *Artificial Intelligence*, *85*, 321–361.

Holte, R. C., Perez, M., R.M.Zimmer, & MacDonald, A. (1996b). Hierarchical A*: Searching abstraction hierarchies efficiently. In *National Conference on Artificial Intelligence (AAAI)*, pp. 530–535.

Horst, R., & Tuy, H. (1996). *Global optimization: Deterministic approaches.* Springer.

James, G. M., Radchenko, P., & Lv, J. (2009). DASSO: Connections between the Dantzig selector and lasso. *Journal of the Royal Statistical Society, Series B*, *71*, 127–142.

Johns, J., & Mahadevan, S. (2007). Constructing basis functions from directed graphs for value function approximation. In *International Conference on Machine Learning.*

Johns, J., Petrik, M., & Mahadevan, S. (2009). Hybrid least-squares algorithms for approximate policy evaluation. *Machine Learning*, *76*(2), 243–256.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric spaces. In *International Conference on Machine Learning (ICML).*

Kakade, S. M. (2003). *On the Sample Complexity of Reinforcement Learning.* Ph.D. thesis, University College London.

Kautz, H. A., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI).*

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *International Conference on Machine Learning*, pp. 260–268. Morgan Kaufmann.

Kearns, M., & Singh, S. (2002). Near-polynomial reinforcement learning in polynomial time. *Machine Learning*, *49*, 209–232.

Keller, P. W., Manor, S., & Precup, D. (2006). Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *International Conference on Machine Learning.*

Kocsis, L., & Szepesvri, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML).*

Kolter, J. Z., & Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML).*

Korf, R. (1985). Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*(1), 97–109.

Korf, R. E. (1988). Real-time heuristic search. In *National Conference on AI (AAAI)*.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*, 1107–1149.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Langtangen, H. P. (2003). *Computational Partial Differential Equations* (2nd edition). Springer.

Leckie, C., & Zuckerman, I. (1998). Inductive learning of search control rules for planning. *Artificial Intelligence*, *101*(1-2), 63–98.

Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: A framework for self-aware learning. In *International Conference on Machine Learning*.

Litvinchev, I., & Tsurkov, V. (2003). *Aggregation in Large-Scale Optimization (Applied Optimization)*. Springer.

Maei, H., Szepesvari, C., Bhatnagar, S., Precup, D., Silver, D., & Sutton, R. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems 22*, pp. 1204–1212.

Mahadevan, S. (2005a). Proto-value functions: Developmental reinforcement learning. In *International Conference on Machine Learning*.

Mahadevan, S. (2005b). Representation policy iteration. In *Conference on Uncertainty in Artificial Intelligence*.

Mahadevan, S. (2005c). Samuel meets Amarel: Automating value function approximation using global state space analysis. In *National Conference on Artificial Intelligence*.

Mahadevan, S. (2009). Learning representation and control: New frontiers in approximate dynamic programming. *Foundations and Trends in Machine Learning*, *1*(4).

Mahadevan, S., & Maggioni, M. (2005). Value function approximation with diffusion wavelets and Laplacian eigenfuctions. In *Advances in Neural Information Processing Systems*.

Mahadevan, S., Maggioni, M., Ferguson, K., & Osentoski, S. (2006). Learning representation and control in continuous Markov decision processes. In *National Conference on Artificial Intelligence*.

Mangasarian, O. L. (1995). The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization*, *12*, 1–7.

Mangasarian, O. L. (2004). A Newton method for linear programming. *Journal of Optimization Theory and Applications*, *121*, 1–18.

McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *International Conference on Machine Learning (ICML)*.

Mendelssohn, R. (1980). Improved bounds for aggregated linear programs. *Operations Research*, *28*, 1450–1453.

Mendelssohn, R. (1982). An iterative aggregation procedure for Markov decision processes. *Operations Research*, *30*(1), 62–73.

Mercier, L., & Hentenryck, P. V. (2007). Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *International Joint Conference on AI*, pp. 1979–1985.

Minton, S., Knoblock, C., Kuokka, D. R., Gil, Y., Joseph, R. L., & Carbonell, J. G. (1989). PRODIGY 2.0: The manual and tutorial. Tech. rep., Carnegie Mellon University.

Munos, R. (2003). Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, pp. 560–567.

Munos, R. (2007). Performance bounds for approximate value iteration. To appear in SIAM.

Munos, R., & Szepesvari, C. (2008). Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, *9*, 815–857.

Nilsson, N. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw Hill.

Osborne, M. R., Presnell, B., & Turlach, B. A. (2000). A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, *20*, 389–404.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., & Littman, M. L. (2008). An analysis of linear models, linear value function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning*.

Parr, R., Painter-Wakefield, C., Li, L., & Littman, M. (2007). Analyzing feature generation for value-function approximation. In *International Conference on Machine Learning*.

Patrascu, R., Poupart, P., Schuurmans, D., Boutilier, C., & Guestrin, C. (2002). Greedy linear value-approximation for factored Markov decision processes. In *National Conference on Artificial Intelligence (AAAI)*.

Patrascu, R.-E. (2004). *Linear Approximations For Factored Markov Decision Processes*. Ph.D. thesis, University of Waterloo.

Pearl, J. (1984). *Heristics*. Addison-Wesley, Reading, MA.

Petrik, M. (2007). An analysis of Laplacian methods for value function approximation in MDPs. In *International Joint Conference on Artificial Intelligence*, pp. 2574–2579.

Petrik, M., & Scherrer, B. (2008). Biasing approximate dynamic programming with a lower discount factor. In *Advances in Neural Information Processing Systems (NIPS)*.

Petrik, M., & Zilberstein, S. (2007a). Anytime coordination using separable bilinear programs. In *Conference on Artificial Intelligence*, pp. 750–755.

Petrik, M., & Zilberstein, S. (2007b). Average reward decentralized Markov decision processes. In *International Joint Conference on Artificial Intelligence*, pp. 1997–2002.

Petrik, M., & Zilberstein, S. (2008). Learning heuristic functions through approximate linear programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 248–255.

Petrik, M., & Zilberstein, S. (2009). A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, *35*, 235–274.

Pinar, M. C. (1996). Linear programming via a quadratic penalty function. *Mathematical Methods of Operations Research*, *44*(3), 345–370.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, *1*, 193–204.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. *Machine Intelligence*, *8*, 55–72.

Poupart, P., Boutilier, C., Patrascu, R., & Schuurmans, D. (2002). Piecewise linear value function approximation for factored MDPs. In *National Conference on Artificial Intelligence.*

Powell, W. B. (2007a). *Approximate Dynamic Programming.* Wiley-Interscience.

Powell, W. B. (2007b). Approximate dynamic programming for high-dimensional problems. In *Tutorial presented at the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning.*

Puterman, M. L. (2005). *Markov decision processes: Discrete stochastic dynamic programming.* John Wiley & Sons, Inc.

Rasmussen, C. E., & Williams, C. (2006). *Gaussian Processes for Machine Learning.* MIT Press.

Reinefeld, A. (1993). Complete solution of the eight-puzzle and the benefit of node ordering in IDA*. In *International Joint Conference on AI*, pp. 248–253.

Rintanen, J. (2000). An iterative algorithm for synthesizing invariants. In *National Conference on Artificial Intelligence (AAAI).*

Rogers, D. F., Plante, R. D., Wong, R. T., & Evans, J. R. (1991). Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, *39*(4), 553–582.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence A Modern Approach* (2nd edition). Prentice Hall.

Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica*, *65*, 487–516.

Sacerdott, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, *5*(2), 115–135.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229.

Sanner, S., Goetschalckx, R., Driessens, K., & Shani, G. (2009). Bayesian real-time dynamic programming. In *Intenational Joint Conference on Artificial Intelligence (IJCAI)*.

Scherrer, B. (2010). Should one compute the temporal difference fix point or minimize the Bellman residual?. In *International Conference on Machine Learning (ICML)*.

Schweitzer, P. J., & Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, *110*, 568–582.

Shetty, C. M., & Taylor, R. W. (1987). Solving large-scale linear programs by aggregation. *Computers and Operations Research*, *14*, 385–393.

Smith, T., & Simmons, R. G. (2006). Focused real-time dynamic programming. In *National Proceedings in Artificial Intelligence (AAAI)*.

Stanley, K. O., & Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, *21*, 63–100.

Stolle, M., & Precup, D. (2002). Learning options in reinforcement learning. *Lecture Notes in Computer Science*, *2371*, 212–223.

Strehl, A. L., Li, L., & Littman, M. (2009). Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, *10*, 2413–2444.

Strehl, A. L., & Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In *International Conference on Machine Learning*.

Strehl, A. L., & Littman, M. L. (2008). An analysis theoretical analysis of model-based interval estimation for Markov decision processes. *Journal of Computer System Science*, *74*(8), 1309–1331.

Sutton, R. S., & Barto, A. (1998). *Reinforcement learning*. MIT Press.

Szepesvari, C., & Munos, R. (2005). Finite time bounds for sampling-based fitted value iteration. In *International Conference on Machine Learning (ICML)*.

Szita, I., & Lorincz, A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural Computation*, *18*(12), 2936–2941.

Thayer, J. T., & Ruml, W. (2008). Faster than weighted a*: An optimistic approach to bounded suboptimal search. In *International Conference on Automated Planning and Scheduling*.

Trick, M., & Stanley, E. (1993). A linear programming approach to solving stochastic dynamic programs..

Trick, M. A., & Zin, S. E. (1997). Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics*, *1*(1), 255–277.

Vakhutinsky, I. Y., Dudkin, L. M., & Ryvkin, A. A. (1979). Iterative aggregation–a new approach to the solution of large-scale problems. *Econometrica*, *47*(4), 821–841.

Valtorta, M. (1984). A result on the computational complexity of heuristic estimates for the A* algorithm. *Information Sciences*, *34*, 48–59.

Vanderbei, R. J. (2001). *Linear Programming: Foundations and Extensions* (2nd edition). Springer.

Vapnik, V. (1999). *The Nature of Statistical Learning Theory*. Springer.

Williams, R. J., & Baird, L. C. (1994). Tight performance bounds on greedy policies based on imperfect value functions. In *Yale Workshop on Adaptive and Learning Systems*.

Wolpert, D. (1996). The lack of apriori distinctions between learning algorithms. *Neural Computation*, *8*, 1341–1390.

Yang, F., Coulberson, J., Holte, R., Zahavi, U., & Felner, A. (2008). A general theory of assitive state space abstraction. *Journal of Artificial Intelligence Research*, *32*, 631–662.

Yao, Y., & Lee, Y. (2007). Another look at linear programming for feature selection via methods of regularization. Tech. rep. 800, Ohio State University.

Yoon, S., Fern, A., & Givan, R. (2008). Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, *9*, 638–718.

Zhang, Z., Sturtevant, N. R., Holte, R., Schaeffer, J., & Felner, A. (2009). A* search with inconsistent heuristics. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning. *AI Magazine*, *24*(2), 73–96.

Zipkin, P. H. (1977). *Aggregation in Linear Programming*. Ph.D. thesis, Yale.

Zipkin, P. H. (1978). Bounds on the effect of aggregating variables in linear programs. *Operations Research*, *28*(2), 403–418.