Solving for Lipschitz Continuous Value Functions through Induced MDPs

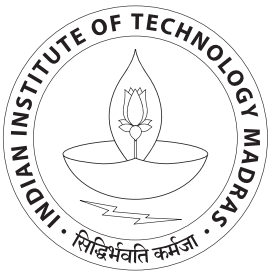
A Project Report

submitted by

DHANVIN HEMANT MEHTA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
&
MASTER OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2015

THESIS CERTIFICATE

This is to certify that the thesis entitled **Solving for Lipschitz Continuous Value Functions through Induced MDPs**, submitted by **Dhanvin Hemant Mehta**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology** and **Master of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Balaraman Ravindran
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Ronald Parr
Research Guide
Professor (Chair)
Dept. of Computer Science
Duke University

Place: Chennai

Place: Durham,NC

Date:

ACKNOWLEDGEMENTS

I am very grateful to Professor Ronald Parr for his constant support, guidance and inspiration. My exploration on Lipschitz continuous value functions started off with an hour long informal discussion with him during my visit to Durham. My collaboration with him has taught me far more than I had anticipated or even hoped for. Always standing by me through the ups and downs of this project, he has been a great mentor and more importantly, a friend.

I would like to thank Professor Balaraman Ravindran, my project guide, for introducing me to reinforcement learning and for making my visits to the department more interesting ever since. Not only has he supported me throughout, but he has gone well out of his way to help me on several occasions. His constant encouragement and insights have been instrumental in making this project possible.

Through these past five years and especially through the last year, my colleagues have kept me sane and happy. My friends - Akhilesh Godi, Karthik Kannan, Chinmay Bapat and Abhik Mondal have helped me take my mind off stressful situations and I have had several fruitful discussions with them. I am especially grateful to Navyata Sanghvi for her constant encouragement throughout. Her interest in this project and her company made the tedious hours spent at the lab enjoyable. Her inputs have been crucial in the formation of this report.

Lastly, collaborating with Jason Pazis and Mark Nemecek from Duke University has been enjoyable. Having worked with Lipschitz continuity before, Jason's experience was essential in

maintaining an interesting direction of research. Mark has been great to work with. Apart from replicating the ball balancing experiments at Duke, he has provided several important insights. His support and dedication has motivated me through this project.

ABSTRACT

A long standing goal in artificial intelligence has been to develop rational agents that can reliably interact with the environment and perform interesting tasks. While traditional control theory enables us to efficiently develop agents for environments whose dynamics can be modelled with reasonable accuracy, in many real world problems the dynamics of the environment is unknown to the agent due to the inherent complexity of the environment. A common approach is to model this problem as a Markov Decision Process (MDP). In reinforcement learning (RL) algorithms, the agent explores the environment to gain knowledge about it while simultaneously trying to maximize the accumulated reward it receives.

Large scale RL problems with continuous state-action spaces pose several challenges. They require generalizing the experience obtained by the agent by tedious feature selection for function approximation. For real-time applications, the RL algorithm must also ensure efficient action selection. Lastly, they must be computationally efficient. This project investigates and extends a recent algorithm designed to address this issue.

Non-parametric approximate linear programming (NP-ALP) is a sample-based approach to value function approximation for MDPs with continuous state- and action-spaces, which requires only a distance function and an estimate of the Lipschitz constant of the value function. Using these inputs and a set of training data, NP-ALP finds a value function that is consistent with the specified Lipschitz constant. Previous theoretical work provided appealing sample

complexity and approximation error bounds for this technique, but the method required solving a linear program in which the number of constraints scaled quadratically with the number of training samples. We decouple the underlying optimization task from the linear programming solution. In doing so, we propose a framework to incorporate arbitrarily local Lipschitz continuity constraints into the traditional MDP resulting in an 'induced MDP'. This framework opens up the possibility of developing new and more efficient approaches for achieving the same objective. We present dynamic programming approaches - a variation on the value iteration and policy iteration algorithms - whose fixed point is the same as the NP-ALP solution. These contributions greatly lower both conceptual and resource barriers to achieving optimal Lipschitz continuous value functions. We show dramatic speed-ups relative to NP-ALP on synthetic problems over a specialized constraint generation approach solved using a state-of-the-art linear program solver. Additionally, we present an efficient homotopy-based approach to find value functions constrained by successively tighter Lipschitz constants without having to recompute the value function each time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

γ	The discount factor
π	The agent's policy
$\mathcal{A}$	action space
a	an action
B	Bellman operator
$\tilde{B}$	approximate Bellman operator
L	Lipschitz constant
L^f	Lipschitz constant for the Lipschitz continuous function f
P	transition probability matrix
Q	state-action value function
Q^*	optimal state-action value function
Q^π	state-action value function following policy π
$\mathcal{R}$	reward distribution
$R(s,a)$	expected reward for state-action (s,a)
r	immediate reward (a sample from the reward distribution)
$\mathcal{S}$	state space
s	a state
t	time-step (used for agent-environment interaction or for indexing a sequence)

V	state value function
V^*	optimal state value function
V^π	state value function following policy π
π_V	greedy policy w.r.t value function V

Batch RL

(s, a, r, s')	A sample obtained by agent-environment interaction
s'	next state
$\mathbb{X}$	A batch of samples $\left\{(s_i, a_i, r_i, s'_i)\right\}_{i=1}^{\infty}$
$\mathcal{N}_k(s)$	k-nearest neighbours of state s for the batch of samples and the distance function being considered.

Part I

Preliminaries

CHAPTER 1

Introduction

Reinforcement Learning (RL) is a widely studied sub-branch of machine learning. It subsumes a wide variety of interesting problems. In its general setting, RL deals with the problem of building an agent that learns to act rationally in an unknown environment by interacting with it, receiving feedback in the form of rewards. The most interesting RL domains (environments) have high dimensional state-action spaces. For several tasks such as helicopter control, a continuous action-space is essential for good control as discrete action spaces are insufficient and can lead to damage of physical components due to jerky control. Conventional algorithms that deal with large state-action spaces employ generalization techniques that give performance guarantees assuming some form of continuity in the value function. We develop the framework for incorporating continuity directly into RL algorithms allowing us to deal with high dimensional continuous state-action spaces.

1.1 Learning smooth value functions

Non-parametric approximate linear programming (NP-ALP) [Pazis and Parr, 2011] is a batch reinforcement learning method that has several significant benefits over traditional approximate linear programming and, generally, over parametric methods. As a non-parametric method, it does not require computationally costly or human labor-intensive feature selection. NP-ALP can be used with continuous actions and has significantly stronger performance guarantees than feature-based ALP. Further, NP-ALP gives strong lower bounds on the rate of convergence and

upper bounds on performance loss using a finite number of samples, even when using noisy, real world samples [Pazis and Parr, 2013*b*].

Despite these strong performance guarantees, several limitations have hindered the adoption of this approach. The method requires solving a linear program where the number of constraints scales quadratically with the number of samples, necessitating clever (though only partially effective) constraint generation techniques to avoid bogging down the LP solver. Another limitation was conceptual: the solution was tightly coupled to the optimization method. The objective function of the LP (minimizing the sum of the state values) did not give insights into the underlying problem that was solved.

Additionally, NP-ALP enforces global Lipschitz continuity (elaborated in 2.1), leading to undesirable properties. Global Lipschitz continuity is beneficial only if the distance metric is *sound* globally. Normally this is not the case. In fact, states normally lie within a *manifold* embedded inside feature-space. Generally, we have a much better handle on distances (similarities) between states that are close to each other and we can use this information to approximate geodesic distances. As a result, local continuity allows for increased robustness and avoids overfitting the value function.

1.2 Our Contributions

We decouple the underlying problem that NP-ALP solves from the optimization method. We introduce the notion of an *induced MDP* derived from a finite number of collected samples and a Lipschitz constant L which determines the degree of continuity imposed on the optimal value function for the induced MDP. The induced MDP can be thought of as having an augmented action-space which, in addition to the original MDP actions, includes actions which raise the state-value in order to comply with the Lipschitz continuity requirements. Since the optimal

value function of the induced MDP is the same as the solution for NP-ALP, it inherits all of NP-ALP's performance guarantees. We show that the induced MDP is very similar to a traditional MDP, allowing us to introduce value iteration and policy iteration algorithms. The policy iteration algorithm is both time and memory efficient and, empirically, yields dramatic speedups over NP-ALP. The analysis and algorithms presented herein make it easier to implement solvers for Lipschitz continuous value functions and to run them on large data sets.

Most importantly, the construction of an induced MDP allows us to seamlessly incorporate Lipschitz continuity locally. This achieves the much-needed flexibility and robustness that NP-ALP lacks.

For clarity and perspective, we emphasize some points which NP-ALP does *not* address. We also do not claim to address these in this thesis. NP-ALP is not a substitute for a parametric method with a carefully chosen set of features - if it is indeed possible to find such a set. NP-ALP does not address exploration. NP-ALP does not solve the curse of dimensionality. The number of samples required to maintain constant maximum gap in the sample set will grow exponentially with the dimension of the state space. This problem is not unique to NP-ALP.

1.3 Organisation of the thesis

1. Chapter 2 introduces the general framework of a Markov Decision Process, explains the key challenges in reinforcement learning (RL) and talks about important classes of algorithms for reinforcement learning.

2. We place this thesis with related work in chapter 3 and introduce non-parametric approximate linear programming (NP-ALP), which forms the basis for this thesis.

3. Our contributions begin in Part II with chapter 4. We develop the theoretical framework of *induced MDPs* allowing us to seamlessly integrate Lipschitz continuity constraints into the value function. We design an operator which provably converges to a unique fixed point - the optimal Lipschitz continuous value function.

4. In chapter 5, we introduce policy iteration and value iteration algorithms that supplement the NP-ALP algorithm, allowing us to solve for better value functions more efficiently. The simplicity of these algorithms promotes widespread adoption of this technique. With an arsenal of efficient algorithms at our disposal, we revisit the induced MDP, and explore interesting variations (settings) for this framework giving us fine control over the the bias we impose.

5. The induced MDP works on a pre-decided Lipschitz constant. In chapter 6, we discuss the effect of the Lipschitz constant L on the induced MDP and present an efficient *homotopy-based* algorithm to move between fixed points of similarly constrained induced MDPs.

6. We then present experimental results for the performance of this framework on standard RL domain - the inverted pendulum and the mountain car - as well as a physical 1D ball balancing problem in chapter 7. Additionally we show some properties of the density of induced MDPs w.r.t the Lipschitz continuity parameter L.

7. Lastly, we discuss how this framework could be used to solve more interesting problems like the helicopter domain and 2D ball balancing, as well as future directions in chapter 8.

CHAPTER 2

Background

2.1 Lipschitz Continuity

A function f is Lipschitz continuous if $\exists$ a Lipschitz constant L such that it satisfies the following constraint for every two elements x and y in its domain:

$$|f(x) - f(y)| \leq Ld(x, y)$$

where d is a distance function.

Every Lipschitz continuous function is continuous at any point x. Given:

- Any $\epsilon > 0$, and
- A Lipschitz continuous value function f with Lipschitz constant L,

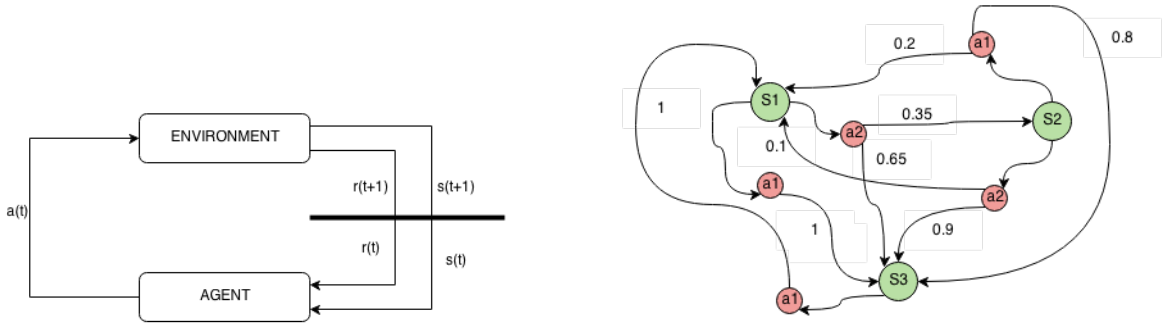
Let y be any point s.t. $d(x, y) < \frac{\epsilon}{L} = \delta$. Then, $|f(x) - f(y)| \leq Ld(x, y) < \epsilon$.

The converse is not true. An example of a function that is continuous but not Lipschitz continuous is the function $f(x) = \sqrt{x}$ in the domain $x \in [0, 1]$. Since the derivative of $f'(x) = \frac{1}{2\sqrt{x}} \to \infty$ as $x \to 0$, the function becomes infinitely steep and no Lipschitz constant would work.

Thus, Lipschitz continuity is stricter than regular continuity.

2.2 Markov Decision Processes

A general problem in AI is that of an agent learning to act rationally in an environment by interacting with it and receiving periodic feedback in the form of 'rewards'. The agent's objective is to learn to maximize the expected cumulative reward it receives from the environment and, thus, to learn a good policy for a desired task. The distinction between the agent and the environment is made based on what can be controlled. If we treat the agent and the environment as a dynamic system, the agent is the part of the system that can be controlled while the environment is the part of the system that cannot be controlled.



(a) The agent-environment interface

(b) An simplistic MDP with 3 states and 2 actions.

Figure 2.1: (a) Illustrates the agent-environment interface. At every time step, the agent senses its state and a reward from the environment, and takes an action according to the state which it finds itself in. (b) A simple MDP with three states - $S1, S2$ and $S3$ and two actions $a1, a2$. The transition probabilities are encoded as edge probabilities (For example: taking action $a1$ in $S2$ causes the agent to move to state $S1$ with probability 0.2 and to $S3$ with probability 0.8). The rewards are excluded from the figure for clarity.

Definition 1. *A Markov Decision Process (MDP) models such a dynamic system as a 5-tuple* $(\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma)$*, where*

1. $\mathcal{S}$ *is the state space of the process - the set of all possible states an agent can find itself in*
2. $\mathcal{A}$ *is the action space - the set of all possible actions an agent can take.* [1]

[1] While it is true that this set may depend upon the state, in general this constraint can be imposed by appropriately composing the transition probabilities and rewards for the forbidden state-action pairs

3. *P is a Markovian transition model $P(s'|s,a)$ denotes the probability of a transition to state s' when taking action a in state s*

4. *$\mathcal{R}$ is the underlying reward distribution of the environment. At any given instant, the reward is drawn from an underlying unknown conditional reward distribution given the state s, the action a and the next state s'. $R(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)\mathbf{E}\big[\mathcal{R}(s,a,s')\big]$. Thus, $R(s,a) = \mathbf{E}_P\big[\mathcal{R}(s,a)\big]$ is the expected reward obtained from the environment upon taking action a in state s.*

5. *$\gamma \in (0,1)$ is a discount factor for future rewards.*

Definition 2. *A deterministic policy π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice in state s. A non-deterministic policy, on the other hand is a conditional distribution over actions given the current state.*

As the agent interacts with the environment, it experiences samples of the form $(s,a,r,s')_{t=1}^{\infty}$ where

1. s is the current state of the agent
2. a is the action taken by the agent knowing that it is in state s
3. s' is the next state the environment puts the agent in
4. r is the reward the environment gives the agent for taking action a in state s

The discounted cumulative reward it receives from the environment (R_∞):

$$R_\infty \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} \gamma^{t-1} r_t$$

Since the environment is stochastic, an optimal agent should discover a policy π^* that would maximize $\mathbf{E}_\pi^*\big[R_\infty\big]$

2.2.1 A mouse in a maze

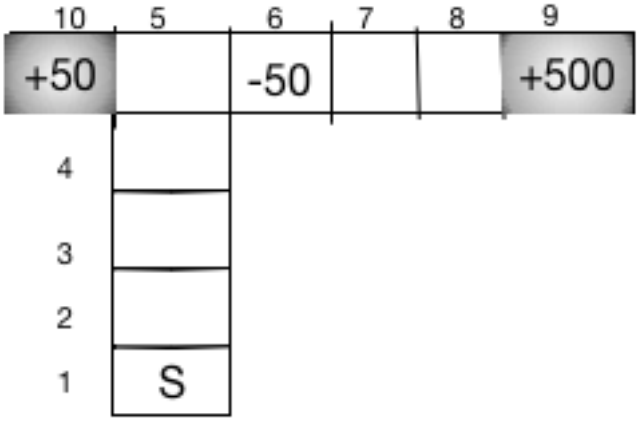


Figure 2.2: A small deterministic MDP modelling the dynamics of a mouse in a maze. 'S' represents the start state and the numbers represent the immediate rewards. State 10 has a small piece of cheese while state 9 has a big piece of cheese. Once the mouse enters either of these states, the episode terminates.

Figure 2.2 models a mouse in a maze as an MDP. There are 10 states and the mouse starts from state 1 (marked S) and can take an action to move to a neighbouring state. States $6, 9, 10$ have non-zero rewards as shown in the figure. All other states have 0 rewards. All rewards are deterministic and depend only on the landing state - s'.

Let's first assume that all transitions are deterministic.

- If the mouse chooses to enter terminating state[2] 10 from state 5, it would obtain $R_\infty = 50$.
- If on the other hand, it chooses to go right and then go on, it would receive a cumulative reward of $R_\infty = -50 + \gamma(0) + \gamma^2(0) + \gamma^3(500)$.
- If the discount factor is very low $\gamma = 0.1$, a rational mouse will use a short-sighted policy and will turn left at state 5 and reap the immediate $+50$ reward instead of braving the rough terrain in state 6 to obtain the larger piece of cheese.
- If on the other hand, $\gamma = 0.98$, the mouse will follow a far-sighted policy and opt to brave the rough terrain.
- For most problems of interest, far-sighted policies need to be learnt and the discount factor is set high.

[2] finite episodes can be modelled with the help of an extra absorbing state giving 0 rewards.

Let's now assume non-deterministic transitions.

- Imagine non-deterministic transitions in state 8 which push the mouse back to state 5 with some probability.
- The policy of choosing to brave state 6 is more risky and, depending on the chance of failure, the mouse may end up choosing to take the safe option from state 5.

Planning v/s Reinforcement Learning

Planning in an MDP is the task of finding an optimal policy for an agent [3] given complete knowledge of dynamics of the environment (known P and R). Reinforcement learning, on the other hand, deals with planning in an unknown environment. The agent must explore the environment to gain experience and act based on the knowledge it has accumulated. In this example, a planning problem would be one in which the mouse knows the map, the rewards and the transition probabilities while a reinforcement learning problem would be one in which the mouse has no prior knowledge and must learn an optimal policy while trying out actions to estimate state-transition probabilities and rewards. In fact, one class of RL algorithms explicitly maintains the agent's belief of the environment at any given point of time thus combining function approximation with planning [4].

2.3 Reinforcement Learning

Consider the task of learning to ride a bicycle. The agent here is the learner and can control the steering, the pedal speed, etc. By controlling these, it receives periodic feedback from the environment (falling down, going off-track, losing balance). Note that the agent does not know apriori what reward/next state to expect upon taking an action in a given state. The agent must interact with the MDP, typically observing the state and immediate reward at every step. At

[3] It can easily be shown that there exists an optimal deterministic policy for a stationary MDP
[4] model-based RL

each time step, the agent observes its current state s, chooses an action a, and observes the resulting state s' along with the reward r. This can be essentially looked upon as sampling from the transition probability distribution and the reward function. In general, we can say that the agent learns from samples of the form of $(s, a, r, s')_{t=1}^{\infty}$.The goal of the agent is to find a mapping from states to actions (a policy π) such that it maximizes the expected cumulative reward it receives from the environment (R_{∞}).

The transition model P and the reward function R unknown to the learner and must be estimated either implicitly (model-free RL [Vlassis and Toussaint, 2009]) or explicitly (model-based RL [Kuvayev and Sutton, 1997]). Reinforcement learning algorithms can be online or offline (batch RL) [Lagoudakis and Parr, 2003].

Challenges in RL

This added burden of learning the environment 'on the fly' poses special challenges in reinforcement learning. A successful RL algorithm must deal with the following three challenges simultaneously -

1. **Sequential Decision Making**: An action taken in the current time step affects not only the immediate reward obtained by the agent, but also the next state it transitions to. This implies that an action is responsible for the future the agent sees. Thus, the agent needs to strike a balance between a good immediate reward and reinforcements expected in the future.
2. **Exploration/ Exploitation Dilemma**: Imagine the learning agent has gained some knowledge of the environment through its interactions. Now, the more knowledge it has gathered, the better it can *exploit* it to gain better cumulative rewards. However, in order to gain more knowledge, it must purposely try out actions which could be sub-optimal (*explore*). A purely exploiting agent with no knowledge is as bad as a purely exploring agent who never cares for utilizing what has been learnt. This dilemma is the exploration/exploitation dilemma or the problem of dual control. [Feldbaum, 1960]
3. **Generalization**: This is the problem pervasive in machine learning - how to generalize across states or state-actions. In order to build statistically robust or even scalable algorithms, some kind of model fitting or function approximation is inevitable.

It must be emphasised here that the real complexity in reinforcement learning stems from

having to solve these challenges *simultaneously*. For instance, the exploration/exploitation dilemma is present even in active learning [Settles, 2010], but it is significantly harder when combined with sequential decision making.

Applications of RL

Reinforcement learning has numerous applications ranging from robotics [Kim *et al.*, 2003], [Ng *et al.*, 2006] to control problems like game playing (the automated checkers player [Samuel, 2000], the game of Go [Silver *et al.*, 2007] and the famous TD-Gammon player [Tesauro, 1995] which achieved a level of play just under the best human players of the time. Additionally, it led to advances in the theory of correct backgammon play by exploring strategies that humans had not pursued.), networks problems like routing packets [Boyan and Littman, 1994], sequential decision problems such as elevator dispatching [Barto and Crites, 1996], algorithm selection [Lagoudakis and Littman, 2000], parameter selection [Ruvolo *et al.*, 2009], discrete optimization problems such as job-shop scheduling [Zhang and Dietterich, 1995], dialog systems [Williams, 2007] and stochastic operations research problems like inventory management systems [Mehta and Yamparala, 2014] and channel allocation [Singh and Bertsekas, 1997]

RL Algorithms

The fundamental classes of RL algorithms can be understood from a planning perspective. In much of the material in this section, we will assume known dynamics (R and P).

Notice that the objective function $\mathbb{E}\left[R_\infty\right]$ is not well tied to the decision variable (a policy). The notion of *value functions* gives us the machinery to tie these two together. They are foundational to a wide range of popular RL algorithms. We show how the objective function, when formulated in terms of the value function, allows us to directly relate it to the policy.

Definition 3. *A state value function is a mapping $V : \mathcal{S} \mapsto \mathbb{R}$ from states to the set of real numbers. The value $V^\pi(s)$ of a state s under a policy π is defined as the expected, total,*

discounted reward when the process begins in state s and all decisions are made according to policy π.

$$V^{\pi}(s) \stackrel{def}{=} \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s\big] \tag{2.1}$$

$$= R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s'\big] \tag{2.2}$$

$$= R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \tag{2.3}$$

Definition 4. *A state-action value function is a mapping $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ from state-action pairs to the set of real numbers. The value $Q^{\pi}(s, a)$ of a state-action pair (s, a) under a policy π is defined as the expected, total, discounted reward when the process begins in state s, the agent takes action a in the first time-step and all decisions (actions) are made according to policy π.*

$$\begin{aligned} Q^{\pi}(s, a) &\stackrel{def}{=} \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s, a_1 = a\big] \\ &= R(s, a) + \gamma \sum_{s'} P(s'|s, a) \mathbf{E}_{\pi}\big[R_{\infty}|s_1 = s'\big] \\ &= R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi}(s') \\ &\quad where\ V^{\pi}(s) = Q^{\pi}(s, \pi(s)) \end{aligned}$$

Having introduced the notion of value functions, we directly formulate the objective function in terms of the value function such that we can implicitly obtain a policy. In order to do this,

we first introduce the Bellman operator B.

$$BV(s) \stackrel{\text{def}}{=} \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \tag{2.4}$$

$$BQ(s,a) \stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{2.5}$$

$$\stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \tag{2.6}$$

In this setting, the objective of a planning algorithm is to learn the optimal policy π^* for choosing actions in the MDP through the observed samples, which yields the optimal value function.

$$V^* \stackrel{\text{def}}{=} V^{\pi^*} \stackrel{\text{def}}{=} \max_\pi V^\pi \tag{2.7}$$

$$Q^* \stackrel{\text{def}}{=} Q^{\pi^*} \stackrel{\text{def}}{=} \max_\pi Q^\pi \tag{2.8}$$

It can be shown that value function is optimal if and only if it is the fixed point of this Bellman operator and can therefore be defined recursively as follows

$$V^*(s) \stackrel{\text{def}}{=} \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

$$Q^*(s,a) \stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

$$\stackrel{\text{def}}{=} R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

or equivalently,

$$V^* = BV^*$$

$$Q^* = BQ^*$$

Thus, solving an MDP is equivalent to solving for the optimal state value function V^* or state-action value function Q^*.

This problem can be solved via dynamic programming [Bellman, 1956] or linear programming [de Farias and Van Roy, 2003] as described later in this section. For small state-action spaces, it is computationally tractable to find the optimal policy, but these problems are uninteresting from a practical viewpoint. Larger, more interesting problems, require certain approximations [Li, 2009] to restore tractability. It should be noted that with complete knowledge of the environment, once can make far better approximations than with unknown environments (in the case of reinforcement learning 2.3).

2.3.1 Value Iteration

The value iteration algorithm [Bellman, 1956] is so called because it operates in the value-function space by iteratively applying the Bellman operator to an initialized value function.

The Bellman operator is a contraction i.e. for any two value functions V_1 and V_2,

$$\begin{aligned} ||V_1 - V_2|| &< \epsilon \\ \Rightarrow ||BV_1 - BV_2|| &< \gamma\epsilon \end{aligned}$$

Let V_t be the value function at iteration t. From the contraction property of the Bellman operator, it follows that the sequence $[V_t]_{t\in\mathbb{N}}$ converges to a unique fixed point, which (as discussed in section 2.2), is the optimal value function for the MDP - V^*.

Despite its simplicity, a lot of effort has gone into applying this basic algorithm tractably to large scale problems resulting in a series of techniques for *asynchronous value iteration*. One

Algorithm 1 Value Iteration

Input: Bellman operator B for the MDP
Initialize $V(s) \leftarrow 0$
Initialize $converged = false$
repeat
  $V_{new} = BV_{old}$
  if $\max(|V_{new} - V_{old}|) < \epsilon$ **then**
    $converged = true$
  end if
  $V_{new} = V_{old}$
until $converged = true$
return V_{new}

major insight into improving the efficiency of value iteration is to selectively updating states instead of naively updating every state at every iteration.

1. Typically an MDP has several *unimportant states* where selecting a wrong action does not cause much damage. The agent can get a reasonable policy despite having an inaccurate estimate of the value of such states. [Li *et al.*, 2007]
2. Furthermore, states whose values are close to optimal should not be repeatedly updated. It would be more economical to update states whose values are less accurate.

Popular techniques like real-time dynamic programming (RTDP) [Barto *et al.*, 1995] and prioritized sweeping [Moore and Atkeson, 1993] build on the ideas listed above to efficiently update the value function until convergence.

2.3.2 Policy Iteration

The objective of planning in an MDP is to learn a good policy rather than a good value function. In fact, several value functions all result in the same policy when the agent acts greedily with respect to them. At each iteration, the algorithm finds successively better policies and terminates upon finding the optimal (best) policy.

Definition 5. *A policy π_1 is better than π_2 iff the value function $V^{\pi_1} > V^{\pi_2}$ i.e. $\forall s \in \mathcal{S}$ $V^{\pi_1}(s) > V^{\pi_2}(s)$*

Algorithm 2 Policy Iteration

Input: $M =< \mathcal{S}, \mathcal{A}, P, \mathsf{R}, \gamma >$
Initialize $\pi_1 \in \mathcal{A}^{\mathcal{S}}$ arbitrarily.
for $i = 1$ **to** ... **do**
    Policy evaluation: Evaluate V^{π_t}
    Policy improvement: $\pi_{t+1} \leftarrow \pi_{V^{\pi_t}}$ (greedy policy w.r.t. V^{π_t})
    if $\max(|V^{\pi_{t+1}} - V^{\pi_t}|) < \epsilon_p$ **then**
        break
    end if
end for
return π_t

Unlike value iteration, policy iteration (Algorithm 2) searches through policy space with the aid of a value function. Starting with an arbitrarily initialized policy, the algorithm iterates between two stages at every iteration - *policy evaluation* and *policy improvement*.

At iteration t, policy evaluation estimates the value function V^{π_t}, while the policy improvement step uses the value function obtained from policy evaluation to generate a new policy π_{t+1} which is strictly better than π_t unless π_t is optimal.

Definition 6. *π_V is a greedy policy w.r.t the value function V*

$$\pi_V(s) \stackrel{def}{=} \arg\max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right)$$

$$\pi_Q(s) \stackrel{def}{=} \arg\max_{a \in \mathcal{A}} Q(s,a)$$

It can be shown that if π_{t+1} is the greedy policy w.r.t V^{π_t}, π_{t+1} is a strict improvement over π_t unless $\pi_t = \pi^*$. [Barto, 1998]. Convergence is also guaranteed in the limit $t \to \infty$. Policy iteration [Howard, 1960] works extremely well in practice and the algorithm converges surprisingly fast [Jaakkola *et al.*, 1994 Gosavi, 2004].In fact, it turns out that it is not necessary to evaluate V^{π_t} accurately. *Modified policy iteration* is a generalization of policy iteration which bridges the gap between value iteration and policy iteration. Here, the policy evaluation step

Algorithm 3 Modified Policy Evaluation

Input: Policy π to be evaluated
Initialize V arbitrarily
Initialize $converged = false$
repeat
  $V_{new} = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_{old}(s')$
  if $\max(|V_{new} - V_{old}|) < \epsilon_m$ **then**
    $converged = true$
  end if
  $V_{new} = V_{old}$
until $converged = true$
return V_{new}

approximately evaluates V^{π_t} by a procedure similar to value iteration (Algorithm 3).

2.3.3 Linear Programming

In this section, we introduce the linear programming method for finding the optimal value function for an MDP. Stating the obvious limitations, we introduce extensions of this idea (approximate linear programming) which attempts to solve large scale planning and reinforcement learning tasks but requires the selection of basis functions. This motivates the recent work on non-parametric approximate linear programming by [Pazis and Parr, 2011] (section 3.1), where just by explicitly modelling continuity, it is possible to give theoretical guarantees on the accuracy of the value function learnt for large-scale, noisy MDPs without the need to fix a set of basis functions.

1. Exact Linear Programming

In the exact LP formulation, every state $s \in \mathcal{S}$ is a decision variable. The formulation below forces the solver to find the minimum value function satisfying $V \geq BV$. For the optimal value function, $V^* = BV^*$ and hence, V^* is the optimal solution for the LP. Although the Bellman operator is non-linear, the constraint $V \geq BV$ can be replaced by a set of linear constraints as

follows.

$$
\begin{aligned}
&minimize \sum_{s} V(s) \\
&subject\ to: \\
&(\forall s, a) V(s) \geq R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')
\end{aligned}
$$

For each state, the optimal policy would be the action for which equality holds in the constraints. Note that such an action must exist for every state.

A major drawback of the above formulation is that it is not scalable. It has $|\mathcal{S}| \times |\mathcal{A}|$ constraints and $|\mathcal{S}|$ decision variables .

2. Approximate Linear Programming (ALP)

In real world domains, the state space is too large and often continuous, making exact representation of the state space intractable. In these cases, we solve the planning problem by approximating the value function by a linear combination of basis functions/features. $V(s) = \phi(s)^T w$, where ϕ is the feature vector, and w is the weight vector associated with it. The decision variables are the weights corresponding to each basis function. The objective function also admits *importance weights* - $\rho(s)$ as certain regions of the state space might be more important than others.

The approximate linear program is formulated as :

$$
\begin{aligned}
minimize \quad & \sum_{s} \rho(s) \phi^T(s) w \\
subject\ to: & \\
(\forall s, a)\ \phi^T(s) w \quad &\geq \quad R(s,a) + \gamma \sum_{s'} P(s'|s,a) \phi^T(s') w
\end{aligned}
$$

The approximation drastically reduces the number of decision variables (from the number of states to the number of basis functions (weights)). However, note that it does not reduce the number of constraints. One can reduce the number of constraints through sampling. If we make some assumptions about the sampling distribution [de Farias and Van Roy, 2003], or if we include regularization [Taylor and Parr, 2009], we can bound the chance of violating an unsampled constraint and give performance guarantees.

Recovering the policy from the learnt value function is no longer straightforward due to the interdependence of states. Typically, recovering a policy involves using a model to estimate the Q-value function given the state value function. Another cause of concern is that the state-relevance weights now influence the solution, capturing the trade-off in the approximation quality across various states [De Farias and Van Roy, 2004].

CHAPTER 3

Related Work

Some major limitations of approaches based on approximate linear programming are that they require features, assume discrete actions, and make weak guarantees. Using L_1-norm regularization on the feature weights, Regularized approximate linear programming (RALP) Petrik *et al.*, 2010 is a parametric method that can select a good set of features from a potentially very large pool. Some non-parametric approaches include variable resolution grids [Munos and Moore, 2002] and kernelized methods [Taylor and Parr, 2009]. Fitted Q-Iteration [Ernst *et al.*, 2005] is a batch-training version of the popular Q-Learning algorithm, which has been shown to be very successful in combination with a tree ensemble technique called extremely randomized trees. When compared to ALP based approaches it has the advantage of being robust to noisy samples. On the other hand, compared to the non-parametric ALP, the amount of information that needs to be stored is a multiple (on the order of 50) of the number of samples instead of a small fraction and as with all Q-value based approaches it is unable to deal with large action spaces very well. Non-parametric approaches abound, e.g., Farahmand *et al.*, 2009 but these also assume discrete actions and rely upon difficult to estimate quantities, such as concentrability coefficients, in their bounds.

Lipschitz continuous value functions naturally arise in a number of important domains in reinforcement learning literature. Hinderer, 2005 presents tools for analyzing Lipschitz continuity of the value functions in MDPs. Non-parametric approximate linear programming (NP-ALP) proposed by Pazis and Parr [Pazis and Parr, 2011] [Pazis and Parr, 2013*b*] circumvent the need for feature selection by imposing Lipschitz continuity directly into the value function. The degree of continuity enforced acts as a regularizer. If the Bellman operator is applied exactly

at sampled states (by integrating over all possible next states), NP-ALP offers max norm performance guarantees. We refer the reader to the original NP-ALP papers for more details; the precise definitions of the error terms are not complicated but require a fair amount of exposition. If next states are sampled, NP-ALP uses a nearest neighbor averaging scheme that is consistent and provides high probability performance guarantees for finite sample sizes. Our work builds upon the ideas of NP-ALP and addresses several of its shortcomings. We outline NP-ALP later in this section since our work is directly motivated by it.

3.1 Non-parametric approximate linear programming

Section 2.3.3 introduced approximate linear programming (ALP) for the planning problem. The lack of complete knowledge about the environment often leads to ill-formed LPs which yield unbounded solutions. As we shall see, one can readily impose Lipschitz continuity by adding linear constraints to the linear program allowing us to readily adapt ALP to reinforcement learning overcoming the problems mentioned here and in section 2.3.3.

Given a finite batch $\mathbb{X}$ of (s, a, r, s') samples from the underlying MDP, non-parametric approximate linear programming [Pazis and Parr, 2011] formulates a linear program on the set of all states encountered in the sample set, $\overline{S} = s : s \in \mathbb{X}$, by explicitly modeling Lipschitz continuity constraints and approximating the Bellman operator for each state based on the observed sample for that state and samples obtained from the neighbourhood of that state.

In order to be Lipschitz continuous with Lipschitz constant L, for every two states s and s' in $\mathcal{S}$, a value function must satisfy the following constraint :

$$|V(s) - V(s')| \leq Ld(s, s')$$

where $d(s, s')$ is the distance between the two states in some metric space. In practice L acts as a regularizer and will differ from L^{V^*}, either to avoid overfitting by design or due to lack of sufficient knowledge.

Let $\mathcal{F}_L$ denote the set of functions with Lipschitz constant L. The constraint $V \in \mathcal{F}_L$ can then be enforced by a set of linear constraints:

$$(\forall s, s' \in \overline{S}) \;\; V(s) \;\; \geq \;\; V(s') - Ld(s, s')$$

The linear program is formulated as

$$\begin{aligned} &minimize \sum_s V(s) \\ &subject\ to: \\ &(\forall (s,a) \in \mathbb{X}) \quad V(s) \geq \tilde{B}Q(s,a) \\ &V \in \mathcal{F}_L \end{aligned}$$

Here $\tilde{B}$ is an approximation of the true Bellman operator derived from a maximum likelihood estimate of the MDP (P and R) for the given batch of samples. Pazis and Parr give performance guarantees when approximating the Bellman operator as follows

$$\tilde{B}Q(s,a) = \frac{\sum_{(s_i,a_i) \in \mathcal{N}_k(s,a)} R(s_i,a_i,s_i^{'}) + \gamma V(s_i^{'}) - Ld_Q(s,a,s_i,a_i)}{k} \tag{3.1}$$

where $\mathcal{N}_k(s,a)$ is the set of k-nearest neighbours[1] of state-action pair (s,a) among the batch of samples and d_Q is a distance function in some metric space containing state-actions.[2]

[1] One of the nearest neighbours is the state-action pair (s,a) itself.

[2] Note that the Lipschitz constraints are bounding the state value function defined based on d - the distance function in some metric space containing states.

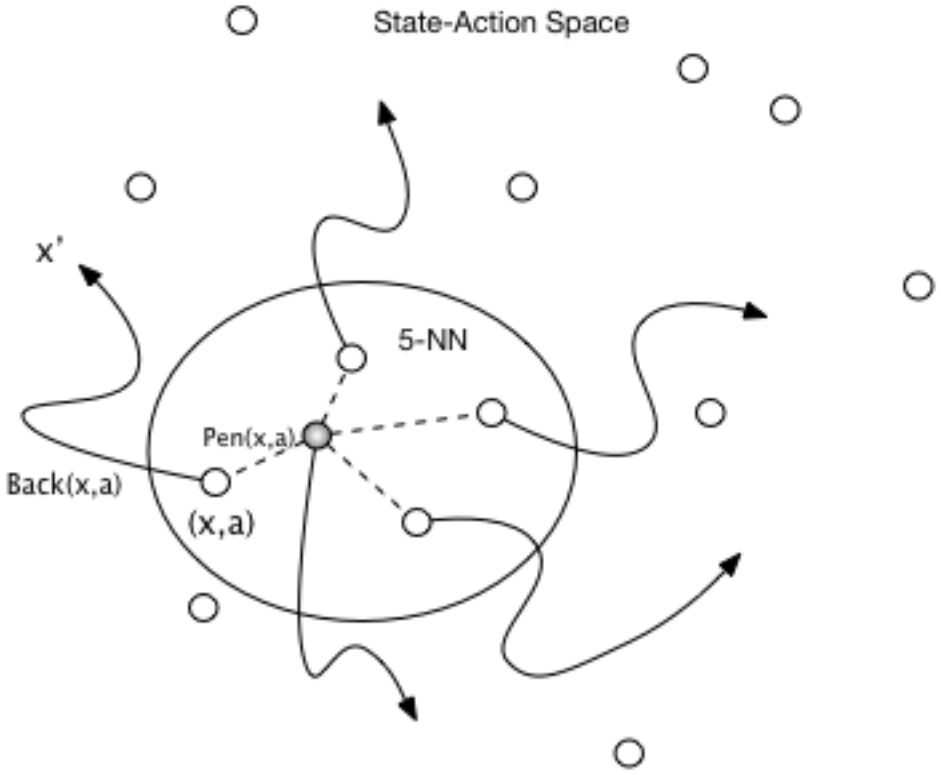


Figure 3.1: The kNN based approximate Bellman operator as proposed by Pazis and Parr, 2013*b* . The figure shows state-actions embedded in a 2D euclidean space. Each state-action corresponds to a sample and is therefore associated with next-state transition and a reward (marked by the zigzag lines) contributing to the backup $\big[Back(x,a) = R(x,a,x') + \gamma V(x')\big]$. The 5 nearest neighbours of the shaded state contribute to the approximate Bellman operator. The penalty is proportional to the distances between the shaded state to its neighbours (marked by the dashed lines) $\big[Pen(x,a) = Ld(s,a,x,a)\big]$.

3.1.1 Properties

NP-ALP has several important properties that make this formulation particularly useful -

1. **Sparseness of LP formulation** - The continuity constraint on the state-value function V is defined globally over the state space and not only for states in $\overline{S}$. However, adding the continuity constraints only for states in $\overline{S}$ is sufficient as the continuity constraints on the remaining states will not have any effect on the solution of the LP. This sparseness enables us to generalize over continuous state spaces using only the sample set. Also note that adding states not in $\overline{S}$ or incorporating importance weights to the objective function will not affect the LP.

2. **Sparseness of solution** - Note that in the NP-ALP formulation (3.1), $\forall s \in \overline{S}$, $V(s)$ is lower bounded either by one of the Bellman constraints or one of the Lipschitz constraints. From the objective function, it is clear that at least one of these constraints must be tight for each state. As pointed out in Pazis and Parr, 2011 , due to triangle inequality (since d is the distance function in a metric space), we only need to store those states which have a tight Bellman constraint in the solution of the LP (Bellman states). This spareness is not only useful for sparsifying the solution, allowing efficient action selection, but also makes evaluation significantly faster and admits optimizations exploiting this property [Pazis and Parr, 2013*b*].

3. **Model-free continuous action selection** - The actions in $\mathbb{X}$ can come from a continuous domain. As discussed, each Bellman state s in the solution of the LP is associated with a unique action (corresponding to the action - $\pi(s)$ - maximizing its Bellman constraints) and a value. Let S_B be the set of Bellman states. Then, given a new state t,

$$\pi(t) = \pi(\arg\max_{s \in S_B} \{V(s) - d(t, s)\}).$$

   This has two important consequences. Firstly, only actions that are encountered in the sample set $\mathbb{X}$ can be selected. Secondly, the complexity of policy execution is independent of the size of the action space. This allows us to deal with continuous, multidimensional action spaces. Of course, in such cases, efficient exploration is required in order to get a good coverage of the state space. Pazis and Parr, 2013*a* adresses which issue.

4. **Well defined solution** - In the case of parametric ALP, a single missing constraint can potentially result in an unbounded LP. NP-ALP, on the other hand, guarantees that the value function is always bounded even when the state-action space is poorly sampled in the batch $\mathbb{X}$. This is due to the global Lipschitz continuity imposed on the value function.

Part II

Learning Lipschitz Continuous Value Functions

CHAPTER 4

Induced MDP

The framework of an induced MDP, constructed from the underlying MDP, the batch of samples (experience) obtained by the agent, and the desired degree of Lipschitz-continuity to be enforced through the Lipschitz constant L allows us to seamlessly integrate Lipschitz continuity into the value function. In this chapter, we formally define this framework and show how NP-ALP is an algorithm that solves a specific instance of the much more general framework of induced MDPs.

1. In section 4.1, we introduce the basic building blocks leading up to the definition of an induced MDP.
2. In section 4.2, we define an induced MDP and the modified Bellman operator $T_{\mathbb{M}}$ associated with it.
3. In section 4.3, we define a valid policy as well as the special structure that it exhibits.
4. Section 4.4 proves that the unique fixed point of the modified Bellman operator for an induced MDP is the optimal value function for the induced MDP and yields the optimal policy.
5. In section 4.5, we define optimal policy execution - i.e. given a new state, how the agent uses its learned policy to take an action.

4.1 Preliminaries

Let $\mathbb{X} = \{(s_i, a_i, r_i, s_i^{'})\}_{i=1}^{N}$ be the batch of samples encountered. Let $\overline{S}$ be the set of all states and A be the set of all actions encountered in $\mathbb{X}$.

We allow for the approximate Bellman operator to be defined over arbitrary state-actions as follows

Definition 7. $\forall s \in \overline{S}$, *the action-set of a state* $Act(s)$ *is the set of chosen actions for state* s *for which the approximate Bellman operator* $\tilde{B}$ *is defined.*[1]

Definition 8. $S^* \subseteq \overline{S}$ *is the subset of states for which the operator is defined for at least one action. Hence,* $s \notin S^*$ *iff* $Act(s) = \emptyset$.

Lipschitz constraints are enforced state-wise as follows:

Definition 9. $\forall s \in \overline{S},\ Lip(s) \subseteq \overline{S}$ *is the chosen set of states which constrain the value of that state -* $V(s)$ *- via Lipschitz continuity.*

Definition 10. *The choice set* $\mathcal{C} = (\,(Act(s), Lip(s)) \mid s \in \overline{S})$ *is an ordered set in which the* i^{th} *element is* $(Act(s_i), Lip(s_i))$. *Thus,* $\mathcal{C}$ *determines the domain over which* $\tilde{B}$ *is defined as well as the extent of Lipschitz continuity enforced.*

Definition 11. *A Lipschitz enforcement graph* $\mathcal{G}$ *given* $(\mathbb{X}, \mathcal{C})$ *is a directed graph* $(\mathcal{V}, \mathcal{E})$ *where the vertex set* $\mathcal{V} = \overline{S}$ *and the directed edge set* $\mathcal{E} = \{s_i \rightarrow s_j | s_i \in \overline{S}, s_j \in Lip(s_i)\}$.

Definition 12. $S^*\text{-}CONN = \{s \in \overline{S} | \exists$ *a path in* $\mathcal{G}$ *from* s *to at least one node* $s' \in S^*\}$. *As we will see later, for the value function of a state to be well defined, it is essential that it belongs to the set* $S^*\text{-}CONN$.

4.1.1 Example

For clarity, we illustrate the concept of a 'Lipschitz enforcement graph' with the help of a simple example. Consider a simple domain where the states and actions are real numbers. One can imagine the state to be the position of the agent and the action to be the speed the agent

[1] The approximation is with the help of the batch of samples $\mathbb{X}$.

sets. The environment is a little noisy. Let the sample set be

$$\mathbb{X} = \begin{cases} (1.2, 2, 0.3, 2.3) \\ (2.3, 1, 0.7, 2.7) \\ (2.7, -0.2, 0.1, 2.5) \\ (2.65, 0, 0.1, 2.75) \\ (-5, 5, 0, -2) \\ (-2, -3, 1.0, -3.4) \end{cases}$$

The set of states occurring in the batch - $\overline{S} = \{1.2, 2.3, 2.7, 2.5, 2.65, 2.75, -2, -3.4, -5\}$. For simplicity we consider the setting where $\forall s \in \overline{S},\ Act(s)$ is the set of actions observed from state s in the batch $\mathbb{X}$ and the Lipschitz constraints are imposed as per the table below. The action sets are listed below -

Choice Sets		
s	$Act(s)$	$Lip(s)$
1.2	$\{2\}$	$\{-2, 2.3, 2.7\}$
2.3	$\{1\}$	$\{2.75\}$
2.7	$\{0.2\}$	$\{-2, 2.65\}$
2.5	$\{\phi\}$	$\{-5, -3.4\}$
2.65	$\{0\}$	$\{1.2, 2.7\}$
2.75	$\{\phi\}$	$\{-2, 1.2\}$
-5	$\{5\}$	ϕ
-2	$\{-3\}$	$\{1.2\}$
-3.4	$\{\phi\}$	ϕ

Table 4.1: Action sets and lipschitz constraints for the induced MDP

Notice in figure 4.1, that the state -3.4 is a hanging state i.e. its value is not constrained by anything. Such states must be pruned out to ensure that the solution is well-defined as in the

case of NP-ALP.[2]. The Lipschitz enforcement graph for $\mathbb{X}$ is shown below (Fig 4.1).

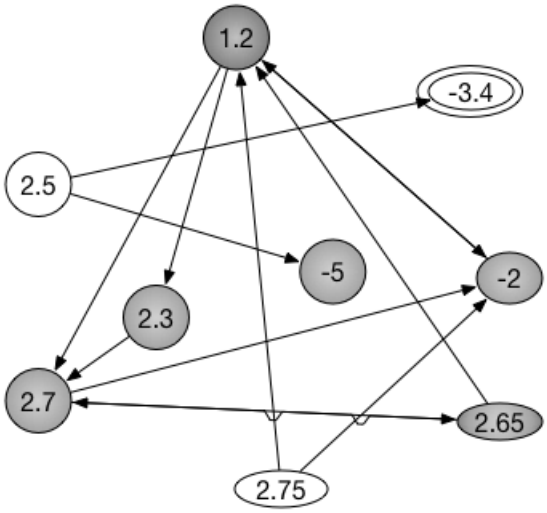


Figure 4.1: The Lipschitz enforcement graph for $\mathbb{X}$. The shaded states $\in S^*$. Note that state -3.4 is not connected to a shaded state and hence must be discarded.
Thus, $S^*\text{-}CONN = \overline{S} \setminus \{-3.4\}$.

4.2 The induced MDP

Definition 13. *Given a batch of samples $\mathbb{X}$ and a Lipschitz constant L, we define an induced MDP $\mathbb{M} =< S, A, P, R, \mathcal{C}, L >$, where $S = S^*\text{-}CONN$ and $A = \bigcup_{s \in S} Act(s)$; P and R are empirically determined from the samples so as to respect the pooling done in the approximate Bellman operator $\tilde{\mathcal{B}}$. $\mathcal{C}$ is derived from $\mathbb{X}$ as defined above.*

We will denote an induced MDP compactly as $< \mathcal{C}, L >$, when comparing between MDPs that differ in these parameters alone.

Definition 14. $\mathcal{C}_1 \succ \mathcal{C}_2 \iff \forall s \in S, Act_2(s) \subseteq Act_1(s) \wedge Lip_2(s) \subseteq Lip_1(s)$ *and*
$\exists s \in S$ *s.t.* $Act_2(s) \subset Act_1(s) \vee Lip_2(s) \subset Lip_1(s)$

Consider $\mathbb{M}_1 =< \mathcal{C}_1, L >$ and $\mathbb{M}_2 =< \mathcal{C}_2, L >$ with $\mathcal{C}_1 \succ \mathcal{C}_2$, then a state in $\mathbb{M}_1$ has at least as many choices as the same state in $\mathbb{M}_2$ in terms of actions and constraints.

[2]NP-ALP does not require such pruning because global lipschitz constraints ensure that $S^*\text{-}CONN = \overline{S}$

Definition 15. *The operator $T_{\mathbb{M}}$ is defined for the induced MDP as follows -*

$\forall s \in \overline{S}$[3],

$$T_{\mathbb{M}}V(s) = max \begin{cases} \max\limits_{a \in Act(s)} \tilde{B}Q(s,a) \\ V(s_i) - Ld(s,s_i) \quad \forall s_i \in Lip(s) \end{cases}$$

Intuition behind $T_{\mathbb{M}}$

The operator $T_{\mathbb{M}}$ is analogous to the Bellman operator B. In addition to maximizing over all actions for which the Bellman operator is defined, $T_{\mathbb{M}}$ incorporates the least value the state must take in order for the function to be Lipschitz continuous.

Like B, $T_{\mathbb{M}}$ operates in function space. However, the fixed point of $T_{\mathbb{M}}$ is a Lipschitz continuous value function unlike the fixed point of B.

4.2.1 NP-ALP solves a specific induced MDP

We now consider NP-ALP [Pazis and Parr, 2013*b*] and show how it readily fits into the more general framework of induced MDPs. In order to do this, we construct an induced MDP $\mathbb{M}$ from the sample set $\mathbb{X} = \{(s_i, a_i, r_i, s_i^{'})\}_{i=1}^{N}$ and subsequently, show that the solution of NP-ALP is a fixed point of the operator $T_{\mathbb{M}}$.

Consider the following induced MDP -

1. **State space** - $S = \overline{S}$
2. **Action sets** - $\forall s \in S$, $Act(s)$ is the set of actions encountered from s in $\mathbb{X}$. Here, S^* is the set of all states in $\overline{S}$ which have any transition in $\mathbb{X}$. If the sample set is obtained from trajectories, these are all but the terminal states of the trajectories.

[3]Note that $S^* \subseteq S$

3. **Lipschitz constraints** - Global Lipschitz constraints are imposed. i.e. $\forall s \in S$, $Lip(s) = \overline{S} \setminus \{s\}$. Due to global continuity, every state $\in S^* - CONN$.

4. **Transition Probabilities** - P is a uniform distribution over the destinations of the k nearest neighbours of the state and 0 elsewhere in $\overline{S}$

5. **Rewards** - $R(s_i, a_i)$ is obtained by averaging $R(s_j, a_j, s_j^{'}) - Ld_Q(s_i, a_i, s_j, a_j)$ over the k nearest neighbours of s_i.

Thus, we have, $\forall (s, a) \in \mathbb{X}$

$$\begin{aligned}
\tilde{B}Q(s,a) &= \frac{\sum\limits_{(s_i,a_i)\in\mathcal{N}_k(s,a)} \left[R(s_i,a_i,s_i^{'}) + \gamma V(s_i^{'})\right] - Ld_Q(s,a,s_i,a_i)}{k} \\
&= \sum_{(s_i,a_i)\in\mathcal{N}_k(s,a)} \frac{r(s_i,a_i,s_i^{'}) - Ld_Q(s,a,s_i,a_i)}{k} + \gamma \sum_{(s_i,a_i)\in\mathcal{N}_k(s,a)} \frac{V(s_i^{'})}{k} \\
&= R(s,a) + \gamma \sum_{s'\in\overline{S}} P(s'|s,a)V(s')
\end{aligned}$$

Now, we qualitatively argue that the solution to NP-ALP is a fixed point of $T_{\mathbb{M}}$ i.e. we show that $V^*_{NP} = T_{\mathbb{M}} V^*_{NP}$ Consider the NP-ALP formulation (section 3.1.1). The objective function ensures that for any given state s, at least one of the constraints lower bounding its value is tight (if it were not the case, the lowering the value of that state would decrease the objective function without violating any other constraints). Hence, the value is the maximum over all the constraining values associated with that state.

The solution of NP-ALP - V^*_{NP} is such that $\forall s \in \overline{S}$[4],

$$\begin{aligned}
V^*_{NP}(s) &= max \begin{cases} \max\limits_{a\in Act(s)} \tilde{B}Q(s,a) \\ V^*_{NP}(s_i) - Ld(s,s_i) \quad \forall s_i \in Lip(s) \end{cases} \\
&= T_{\mathbb{M}} V^*_{NP}
\end{aligned}$$

[4]Note that $S^* \subseteq S$

V^*_{NP} is thus a fixed point of $T_{\mathbb{M}}$. In chapter 4.4, we prove that there is a unique fixed point for this operator. Thus, any algorithm which finds the fixed point for $T_{\mathbb{M}}$ would yield the same solution as NP-ALP.

4.3 Valid policies in an induced MDP

The operator $T_{\mathbb{M}}$ maximizes over the value obtained from approximate Bellman operator as well as over the value obtained by enforcing Lipschitz continuity. Depending on the maximizing expression, the states are partitioned into two sets as follows

- A state s belongs to S_B - the set of Bellman states - iff $\exists a \in Act(s)$ $\tilde{B}Q(s,a) \geq \max_{s' \in Lip(s)} V(s') - Ld(s,s')$. Note that $S_B \subseteq S^*$.
- Else, $s \in S_L$ - the set of Lipschitz states.

Introducing the notion of Bellman trees to characterize valid policies:

Definition 16. *A valid policy graph $\mathcal{G}^\pi$ is a directed forest $\mathcal{G}^\pi = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V} = S$ and the edge set $\mathcal{E} = \{s \to s' \,:\, s \in S_L\,,\, s' \in S\}$ with the constraints -*

1. $\forall s \in S_L \;\; out\text{-}degree(s) = 1$
2. *All trees in $\mathcal{G}^\pi$ are "tight" (defined below)*

By definition, a valid policy graph is a collection of trees, each of whose root is a Bellman state.

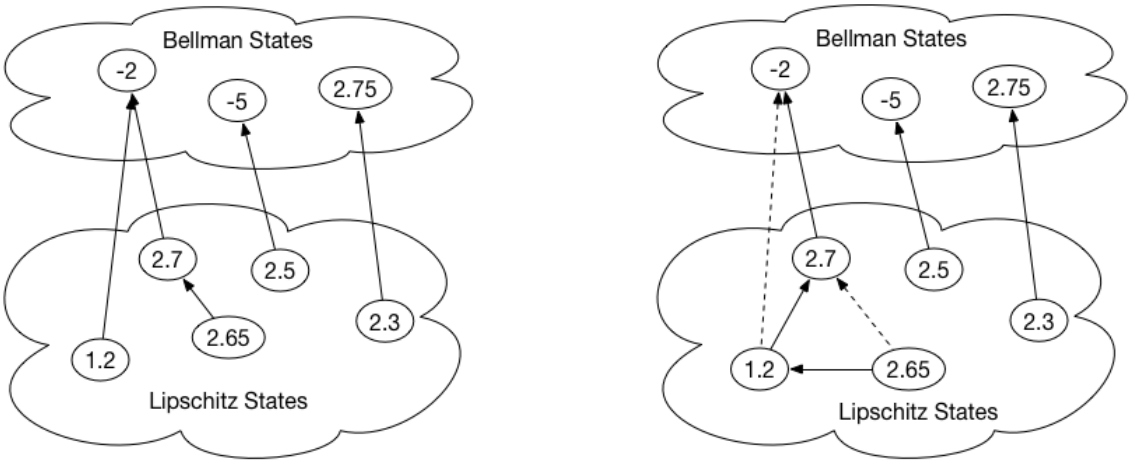
Definition 17. *The Bellman tree at $s \in S_B$ denoted by $T^\pi_B(s)$ is the 'tight' directed tree whose root is s.*[5] *We also define $\forall s \in S_L,\; Out(s)$ to be the node adjacent to s.*

Definition 18. *$\Omega : S_L \to S_B$ is a function which maps a state s in S_L to the Bellman state at the root of its tree in $\mathcal{G}^\pi$. We call Ω the attachment function because it associates the action and value of a Lipschitz state with a Bellman state.*

[5]Note that $\forall s \in S_B$, $\mathcal{G}^\pi$ has a uniquely associated tree rooted at s (a bijection).

Definition 19. *A tight Bellman tree is one where for every node s in the tree, its path to the root has no intersection with $Lip(s) \setminus \{Out(s)\}$.*

Using the simplistic example in section 4.1.1, we illustrate this characterization.



(a) A valid policy graph

(b) An invalid policy graph

Figure 4.2: An illustration of the "tightness" condition for a valid policy graph. (a) is a valid policy graph as the state 2.65 is not constrained by state -2. On the other hand, in (b), $T_B^{\pi}(-2)$ is not tight as 2.7 which directly constraints 2.65 lies on the path between 2.65 to -2. Also, -2 directly constraints 2.65 but 2.65 is connected through 2.7 to it.

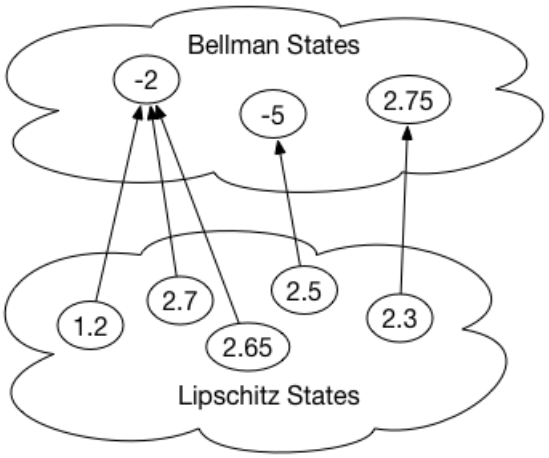
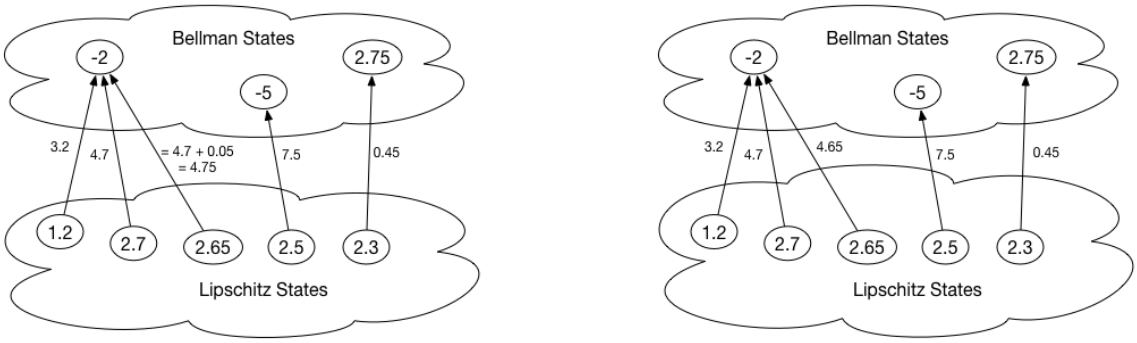


Figure 4.3: For a policy graph in the NP-ALP setting with global Lipschitz continuity to be valid, each Bellman tree must have height at most 2. This is consistent with the characterization of the NP-ALP solution

4.3.1 Effective Distance

Any Lipschitz state s could be looked at as deriving its value from the Bellman state at the root of the tree penalized by an "effective distance" - $\overline{d}$ - equal to the sum of distances along the edges in the path from s to $\Omega(s)$. The notion of $\overline{d}$ is useful in theorem 1 and as shown in Fig 4.4 allows us to view each valid policy graph as a forest of trees, each having height at most 2 (just like in NP-ALP). We will see that this compressed representation can be used in policy execution (section 4.5)



(a) Compressed valid policy graph.

(b) The policy graph for NP-ALP.

Figure 4.4: (a) The compressed policy graph for the local Lipschitz in example 4.1.1. The continuity constraints increases the effective distance between states (encoded here as weights on the edges). Although the distance between state $d(2.65, -2) = 4.65$, the effective distance $\overline{d}(2.65, -2) = 4.75$. This added penalty can be thought of as accommodating uncertainty and penalizing the value function accordingly as explained here.(b) The policy graph for global Lipschitz constraints imposed on the same sample set.

Definition 20. *A valid policy* $\pi =< S_B, \mathcal{A}, \mathcal{G} >$ *is defined as follows -*

$$\pi(s) = \begin{cases} \mathcal{A}(s) & if \quad s \in S_B \\ \pi(\Omega(s)) & if \quad s \in S_L, \end{cases}$$

where $S_B \neq \emptyset \subseteq S^*$, $S_L = S \setminus S_B$; $\mathcal{A} : S_B \rightarrow A$ *is a function with the constraint:* $\mathcal{A}(s) \in Act(s)$. $\Omega : S_L \rightarrow S_B$ *is a function that maps a Lipschitz state to the Bellman state from which*

it derives its action.

Just like a regular MDP, a policy coupled with an induced MDP induces a Markov Chain whose value function is always well defined.

Definition 21. *The value function V^π for a policy π is therefore defined by the system of linear equations -*

$$V^\pi(s) = \begin{cases} R(s, \mathcal{A}(s)) + \gamma \sum_{s' \in S} P(s'|s, \mathcal{A}(s)) V^\pi(s') & \forall s \in S_B \\ V^\pi(Out(s)) - Ld(s, Out(s)) & \forall s \in S_L \end{cases}$$

Theorem 1. *V^π is well defined (bounded) for $L > 0$*

Proof. Consider the policy $\pi =< S_B, \mathcal{A}, \mathcal{G} >$. $\forall s \in S_B$ we have,

$$
\begin{aligned}
V^{\pi}(s) &= R(s, \mathcal{A}(s)) + \gamma \sum_{s' \in S} P(s'|s, \mathcal{A}(s)) V^{\pi}(s') \\
&= R(s, \mathcal{A}(s)) + \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) V^{\pi}(s') \\
&+ \sum_{s'':\Omega(s'')=s'} [P(s''|s, \mathcal{A}(s)) V^{\pi}(s'')] \\
&= R(s, \mathcal{A}(s)) + \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) V^{\pi}(s') \\
&+ \sum_{s'':\Omega(s'')=s'} [P(s''|s, \mathcal{A}(s)) [V^{\pi}(s') - L\overline{d}(s', s'')]] \\
&= R(s, \mathcal{A}(s)) - \gamma \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s)) L\overline{d}(s, s') \\
&+ \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) + \\
&\sum_{s'':\Omega(s'')=s'} P(s''|s, \mathcal{A}(s))] V^{\pi}(s') \\
&= R_{\mathcal{MC}}(s, \mathcal{A}(s)) + \gamma \sum_{s' \in S_B} P_{\mathcal{MC}}(s'|s, \mathcal{A}(s)) V(s')
\end{aligned}
$$

The policy $\pi =< S_B, \mathcal{A}, \Omega >$ defines a Markov Chain $\mathcal{MC}$ whose state space S, transition probability matrix P and reward distribution R are:

$S_{\mathcal{MC}} = S_B$

$P_{\mathcal{MC}}(s'|s) = P(s'|s, \mathcal{A}(s)) + \sum_{s'':\Omega(s'')=s'} P(s''|s, \mathcal{A}(s))$

$R_{\mathcal{MC}}(s) = R(s, \mathcal{A}(s)) - \gamma \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s)) L\overline{d}(s', \Omega(s'))$

$\mathcal{MC}$ is well-defined, as is its value function. □

4.4 Optimality

As of now, we have only introduced the notion of an induced MDP derived from a batch of samples, an underlying MDP and the imposed Lipschitz constant. We now prove useful properties which leads us to the notion of optimality for the induced MDP. In this section, we formalize the objective function of NP-ALP and pave the way for adapting a variety of algorithms for solving traditional MDPs to solve the induced MDP. Later, we use these results to introduce a value iteration and a policy iteration algorithm for solving the induced MDP.

We first prove existence of a fixed point for $T_{\mathbb{M}}$ assuming it converges. We then go on to characterize fixed points of this operator, following which we prove uniqueness of the fixed point.

4.4.1 The existence of a fixed point

Let V_1 and V_2 be value functions defined over a set S.

Definition 22. $V_1 \succ V_2 \iff \forall s \in S, V_1(s) \geq V_2(s)$ *and* $\exists s \in S, V_1(s) > V_2(s)$

Lemma 1. *If* $V_1 \succ V_2$*, then* $T_{\mathbb{M}}V_1 \succ T_{\mathbb{M}}V_2$ *or* $T_{\mathbb{M}}V_1 = T_{\mathbb{M}}V_2$

Proof.

$$
\begin{aligned}
V_1 \succ V_2 \Rightarrow & \forall s, BV_1(s) \geq BV_2(s) \\
& \forall s, s', V_1(s') - L_{\tilde{V}} d(s, s') \geq V_2(s') - L_{\tilde{V}} d(s, s') \\
\Rightarrow & \forall s, T_{\mathbb{M}}V_1(s) \geq T_{\mathbb{M}}V_2(s) \\
\Rightarrow & T_{\mathbb{M}}V_1 \succ T_{\mathbb{M}}V_2 \vee T_{\mathbb{M}}V_1 = T_{\mathbb{M}}V_2
\end{aligned}
$$

□

Definition 23. *A pessimistic value function V s.t. $V \leq T_{\mathbb{M}}V$.*

Theorem 2. *There exists at least one fixed point for $T_{\mathbb{M}}$.*

Proof. Assume the contrary. Let R_{max} be the largest reward obtained in the batch of samples. The value of any state is bounded by $\frac{R_{max}}{1-\gamma}$. Let V_0 be a pessimistic value function. Lemma 1 implies V_0 must either be a fixed point or TV_0 is pessimistic. If $\nexists i$ s.t. $T^i_{\mathbb{M}}V_0$ is a fixed point, then as $i \to \infty$, $(T_{\mathbb{M}})^i V_0 \to \infty$, a contradiction. □

4.4.2 Fixed point characterization

Theorem 3. *A fixed point of $T_{\mathbb{M}}$ corresponds to the value function of a policy $\pi =< S_B, \mathcal{A}, \Omega >$*

Proof. Let V be a fixed point of $T_{\mathbb{M}}$. $S_B = \{s|\ T_{\mathbb{M}}V(s) = BV(s)\}$ and $S_L = \{s|\ \exists s' \in Lip(s),\ T_{\mathbb{M}}V(s) = V(s') - Ld(s,s')\}$. Define $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V} = S$ and the edge set $\mathcal{E} = \{s \to s' \,|\, T_{\mathbb{M}}V(s) = V(s') - Ld(s,s')\}$. Clearly, by construction, $\mathcal{G}$ is a directed forest with every tree rooted in a Bellman state, else there would be a cycle and the values of all states would be unbounded below, while attachment to a Bellman state (all states are S^*-connected) would lead to a bounded value function. The "tightness" of the trees follows from the triangle inequality. □

We now proceed to prove that $T_{\mathbb{M}}$ has a unique fixed point. First, we admit multiple fixed points and introduce the notion of a *least fixed point*. We then go on to show that the *least fixed point* is the only fixed point.

Definition 24. *The least fixed point V^*_{least} is the fixed point for which $\forall$ fixed points $V^* \neq V^*_{least}$, $V^*_{least} \succ V^*$.*

4.4.3 $T_{\mathbb{M}}$ has a unique fixed point

Definition 25. *For a policy $\pi =< S_B, \mathcal{A}, \mathcal{G} >$, a policy-induced MDP $\mathbb{M}^\pi =< \mathcal{C}^\pi, L >$ with the restrictions that*

1. *$Act(s)$ has a null set for all states $s \in S_L$ and the singleton set $\{\mathcal{A}(s)\} \quad \forall s \in S_B$.*
2. *$Lip(s)$ has a null set for all states $s \in S_B$ and the singleton set $\{Out(s)\} \quad \forall s \in S_L$.*

$T_{\mathbb{M}^\pi}$ is the modified Bellman operator for the policy-induced MDP.

Lemma 2. *Consider two MDPs defined as $\mathbb{M}_1 =< \mathcal{C}_1, L >$ and $\mathbb{M}_2 =< \mathcal{C}_2, L >$ with least fixed points V_1^{least} and V_2^{least} respectively. Then, if $\mathcal{C}_1 \succ \mathcal{C}_2$ then $V_1^{least} \geq V_2^{least}$.*

Theorem 4. *$T_{\mathbb{M}}$ has a unique fixed point.*

Proof. Let V_{least} be the smallest fixed point for the induced MDP $\mathbb{M} =< \mathcal{C}, L >$ corresponding to $T_{\mathbb{M}}$ and let V be another fixed point. Now, from theorem 3, we know that each fixed point corresponds to the value function of the corresponding policy. We also know that V would correspond to the value function of a valid policy $\pi =< S_B, \mathcal{A}, \Omega >$. as the system of equations would have a unique solution. We can think of this valid policy as the only (since it well defined - theorem 1) and least fixed point for another induced MDP $\mathbb{M}^\pi =< \mathcal{C}^\pi, L >$ where $\mathcal{C}^\pi$ has a null set for all states $s \in S_L$ and the singleton set $\{\mathcal{A}(s)\} \forall s \in S_B$. Clearly, $\mathcal{C} \succ \mathcal{C}^\pi$. Hence, $V_{least} > V$. However, this is a contradiction. Hence, V_{least} is the only fixed point for $T_{\mathbb{M}}$. □

4.4.4 The optimal Lipschitz continuous value function

Definition 26. *For an induced MDP $\mathbb{M} =< \mathcal{C}, L >$ defining $T_{\mathbb{M}}$, π^* is the policy corresponding to the fixed point and V^* is the fixed point for $T_{\mathbb{M}}$*

Corollary 1. *π^* is the optimal policy for $\mathbb{M}$*

Proof. It has already been proven that $\forall \pi,\ V^{\pi^*} = V^* \geq V^{\pi}$. □

Now that we have established he notion of optimality, we can show that for the induced MDP formulated in section 3.1, NP-ALP proposed by Pazis and Parr finds the optimal value function and policy. In fact, this is evident from the very formulation of NP-ALP.

For the traditional, discrete state-action MDP, with Bellman operator B, the exact linear program formulation which solves for the fixed point is

$$
\begin{aligned}
&minimize \sum_{s} V^*(s) \\
&subject\ to: \\
&(\forall s, a) V^*(s) \geq B V^*(s')
\end{aligned}
$$

For the induced MDP, with the operator $T_{\mathbb{M}}$, the exact linear program formulation which solves for the fixed point is

$$
\begin{aligned}
&minimize \sum_{s} V^*(s) \\
&subject\ to: \\
&(\forall s, a) V^*(s) \geq T_{\mathbb{M}} V^*(s')
\end{aligned}
$$

The optimal value function implicitly defines a valid policy as defined in 3. This is nothing but the optimal policy for the induced MDP (π^*).

4.5 Optimal Policy Execution

The compressed representation of π^* in fig. 4.4 immediately brings out the potential of this framework for continuous model-free action selection. Although the valid policies are defined over the induced MDP's state space, they can be applied over the entire state space of the underlying MDP.

The optimal policy for a new state is defined as follows:

- For an unobserved state, s_{new}

$$\exists a \in Act(s_{new}), \quad \tilde{BQ}(s_{new}, a) \geq \max_{s' \in Lip(s_{new})} V(s') - Ld(s_{new}, s'), \tag{4.1}$$

$$\Rightarrow \pi^*(s_{new}) = \arg\max_{a \in Act(s_{new})} \tilde{BQ}(s_{new}, a) \tag{4.2}$$

- If inequality 4.1 is not true, the action for s_{new} is determined as follows -

$$s^* = \arg\max_{s \in Lip(s_{new})} V(s) - Ld(s_{new}, s)$$
$$\pi^*(s_{new}) = \pi^*(\Omega(s^*))$$

Note that $Lip(s_{new})$ and $Act(s_{new})$ must be computed for a new state. This can be a bottleneck in policy execution especially for real-time applications depending on how Lipschitz continuity is enforced or how the approximate Bellman operator is evaluated. For example, in the case of local Lipschitz continuity within a euclidean ball, the task of evaluating $Lip(s_{new})$ is equivalent to finding all points from a dataset within a particular radius from s_{new}. The approximate Bellman operator we have been using so far involves evaluating the k nearest state-actions.

4.6 Conclusion

In this chapter, we defined the framework of induced MDPs designed to imposed Lipschitz continuity constraints with arbitrary granularity on the learnt value function. In order to compute

the optimal value function, we modified the Bellman operator to incorporate these constraints and proved that the the operator $T_{\mathbb{M}}$ has a unique fixed point. This general framework and the established properties of the operator allow us to introduce a wide range of algorithms and variations for finding the optimal Lipschitz continuous value function for a general induced MDP.

CHAPTER 5

Algorithms

The formulation of induced MDPs allows us to design efficient algorithms as well as have higher flexibility in imposing the Lipschitz continuity bias.

1. Section 5.1 talks about the dynamic programming approaches for solving induced MDPs.
2. Section 5.2 illustrates some interesting types of induced MDPs.

5.1 Dynamic Programming

In this section, we introduce the standard dynamic programming algorithms for finding optimal Lipschitz continuous value functions in a general induced MDP. These techniques (especially policy iteration) are superior as compared to the linear programming approach in terms of efficiency (both time and memory) and ease of implementation. Additionally, it removes the need for bulky LP solvers that have always been a bottleneck for the widespread adoption of linear programming algorithms. These algorithms allow us to find optimal Lipschitz continuous value functions for much larger sample sets than NP-ALP permitted with the added benefit of them being generic for the entire class of induced MDPs. Additionally, they complete the popular arsenal of techniques for planning in MDPs (section 2.2) helping us make a smooth transition from regular MDPs to induced MDPs.

5.1.1 Value Iteration

Algorithm 4 Value Iteration

Input: Induced MDP operator $T_{\mathbb{M}}$, tolerence ϵ
Initialize $V_{old} = V$ pessimistically.
Initialize $converged = false$
repeat
  $V_{new} = T_{\mathbb{M}} V_{old}$
  if $\max(|V_{new} - V_{old}|) < \epsilon$ **then**
    $converged = true$
  end if
  $V_{new} = V_{old}$
until $converged = true$

Having proven convergence to a unique fixed point, we can apply $T_{\mathbb{M}}$ repeatedly from an initial value function to obtain the optimal value function (Algorithm 4).

5.1.2 Policy Iteration

Definition 27. *A greedy policy* $\pi_g(V) = < S_B, \mathcal{A}, \mathcal{T} >$ *for a value function* V *is defined as*

- $S_B = \{s : \exists a \in Act(s), \; T_{\mathbb{M}} V(s) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')\}$
- $\forall s \in S_B, \; \mathcal{A}(s) = \arg\max_{a \in Act(s)} R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$
- $\mathcal{T}$ *is defined by the out-edges of Lipschitz states as follows -* $\forall s \notin S_B, \; Out(s) = \arg\max_{s' \in Lip(s)} V(s') - Ld(s,s')$ *with the restriction that* $\mathcal{T}$ *consists of "tight" trees.*

Algorithm 5 approximately evaluates the value function by repeatedly applying the modified Bellman operator for the policy-induced MDP ($T_{\mathbb{M}^\pi}$). One could alternately carry out policy evaluation by solving a system of linear equations directly.

Theorem 5. *(Policy Improvement) Let* V *be any pessimistic value function for the operator* $T_{\mathbb{M}}$ *s.t.* $V^* \geq V$, V^* *being the fixed point for* $T_{\mathbb{M}}$. *Then,* $V^{\pi_g(V)} \geq V$. *Further, if* $V^{\pi_g(V)} = V$, *then* $V = V^*$.

Algorithm 5 Modified Policy Iteration

Input: Induced MDP operator $T_{\mathbb{M}}$, tolerence ϵ, Evaluation iterations m.
Initialize $V_{old} = V$ pessimistically.
Initialize $converged = false$.
Initialize $\pi = \pi_g(V_{old})$
repeat
  $V_{old} = V^{\pi}$
  for $i = 1$ **to** m **do**
    $V_{new} = T_{\mathbb{M}^{\pi}} V_{old}$
    $V_{old} = V_{new}$
  end for
  $V^{\pi_{new}} = V_{new}$
  if $\max(|V^{\pi_{new}} - V^{\pi_{old}}|) < \epsilon$ **then**
    $converged = true$
  end if
  $\pi = \pi_g(V_{\pi_{new}})$
  $V^{\pi_{old}} = V^{\pi_{new}}$
until $converged = true$

Proof. Consider the induced MDP $\mathbb{M}^{\pi_g} =< \mathcal{C}^{\pi_g}, L >$ corresponding to policy $\pi_g(V)$ and let $\mathbb{M} =< \mathcal{C}, L >$ be the original MDP. Clearly, $\mathcal{C} \succ \mathcal{C}^{\pi_g}$. Let $T_{\mathbb{M}}$ be the operator for $\mathbb{M}$ and $T_{\mathbb{M}^{\pi_g}}$ be the operator for $\mathbb{M}^{\pi_g}$. Since π_g is greedy w.r.t V, $T_{\mathbb{M}^{\pi_g}} V = T_{\mathbb{M}} V$. Since $V^* \geq V$, by definition 22 we have that either $V^* = V$ or $V^* \succ V$.

1. $\mathbf{V^* = V}$
   If $V^* = T_{\mathbb{M}^{\pi_g}} V = T_{\mathbb{M}} V = V$, V is a fixed point for $T_{\mathbb{M}}$ and $T_{\mathbb{M}^{\pi_g}}$. Hence, π_g is the optimal policy and $V = V^*$.

2. $\mathbf{V^* \succ V}$
   $V^* = T_{\mathbb{M}} V = T_{\mathbb{M}^{\pi_g}} V$ or $\Rightarrow V^* \succ T_{\mathbb{M}} V = T_{\mathbb{M}^{\pi_g}} V \succ V$. (from lemma 1)

   (a) $\mathbf{V^* = T_{\mathbb{M}^{\pi_g}} V = T_{\mathbb{M}} V}$
   $T_{\mathbb{M}^{\pi_g}} V$ is a fixed point for $T_{\mathbb{M}}$. We also know that $T_{\mathbb{M}^{\pi_g}} V \succ V$ i.e. V is pessimistic w.r.t. $T_{\mathbb{M}^{\pi_g}}$ and therefore, $V_L^{\pi_g} \geq T_{\mathbb{M}^{\pi_g}} V$. $\mathcal{C} \succ \mathcal{C}^{\pi_g} \Rightarrow V^* \geq V^{\pi_g}$ (lemma 2). Hence, $V^* \geq V^{\pi_g} \geq T_{\mathbb{M}^{\pi_g}} V$. But, $V^* = T_{\mathbb{M}^{\pi_g}} V \Rightarrow V^{\pi_g} = V^*$. Hence, π_g is the optimal policy.

   (b) $\mathbf{V^* \succ T_{\mathbb{M}} V = T_{\mathbb{M}^{\pi_g}} V \succ V}$
   Once again, since V is pessimistic w.r.t. $T_{\mathbb{M}^{\pi_g}}$ and hence, $V_L^{\pi_g} \geq T_{\mathbb{M}^{\pi_g}} V$. Hence, $V_L^{\pi_g} \succ V$

□

5.2 Variations

Having developed the necessary theory for induced MDPs, in this chapter we demonstrate the flexibility of this framework by introducing three important problem settings in this framework and discuss the biases they impose.

Induced MDP Instances		
Instance Name	Scope of Bellman operator	Lipschitz continuity
In-Sample-Global	Observed state-actions	Global constraints
In-Sample-Local	Observed state-actions	Local constraints
Out-Of-Sample	All actions (discrete) for observed states	No explicit constraints

Table 5.1: Different instances of the induced MDP framework. NP-ALP offers a solution to the In-Sample-Global instance enforcing global Lipschitz constraints. In the In-Sample-Local instance, each state is constrained only locally. Out-Of-Sample defines the approximate Bellman operator even for unseen state-actions.

5.2.1 In-Sample

In this setting, the approximate Bellman operator is defined only for observed state-actions while Lipschitz constraints are imposed globally (each state is constrained directly by every other state). This restriction is crucial while dealing with continuous actions. During policy execution, given a new (unobserved) state s, an in-sample method would only require evaluating $Lip(s)$. Without this restriction, we would have to evaluate the Bellman operator for every allowable action from the unseen state, making it intractable for large action spaces.

The two variations we talk about in this section involve local and global continuity.

5.2.2 In-Sample-Global

Global Lipschitz continuity is a very strong bias and unless we have a good estimate of distances, it can result in poor value functions. Another drawback of global continuity is that it is more sensitive to noise. If the Bellman operator is over-estimated for a particular state, it corrupts the value function to a much greater extent than in the case of local constraints.

That said, imposing global constraints definitely helps reduce sample complexity given a good estimate of the distance function. NP-ALP solves exactly this problem setting and provides very good theoretical guarantees on the quality of the value function learnt.

5.2.3 In-Sample-Local (Manifold)

In several cases, we are more certain about local distances between states. One can imagine the states lying on an unknown manifold embedded within the ambient space. Our motivation comes from a common approach to learning manifolds - using local information. As illustrated in figure 5.1, the states sampled lie on some unknown manifold. Imposing global Lipschitz constraints would incorrectly consider the true distance between the extreme states as the ambient distance rather than the geodesic distance. Imposing local Lipschitz continuity on the other hand results in an effective distance $\overline{d}$ - the sum of local distances - which is a much better estimate of the geodesic distance.

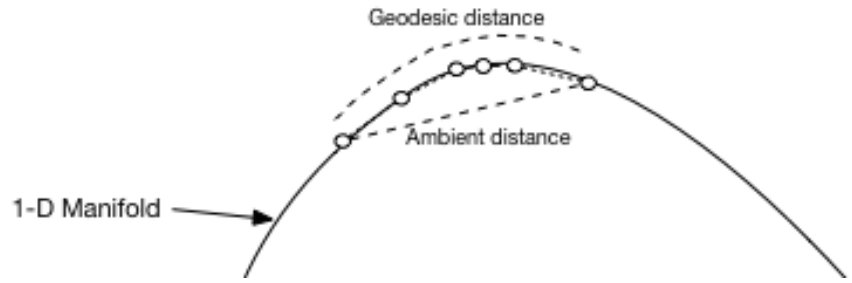


Figure 5.1: A motivating example for local constraints. In the simple case of a 1D manifold embedded in a 2D space, with circles representing states, the curved geodesic distance is better approximated as the sum of local ambient distances (dashed lines between states).

The notion of locality can be arbitrarily chosen. Two standard ways of defining locality are

1. k_l nearest neighbours [1] of a state.
2. an ϵ euclidean ball around the state.

The extent of continuity enforced by each of the aforementioned definitions depends largely (albeit in different ways) upon the sample density in the state-space. While the k_l nearest neighbours approach localizes the continuity to smaller regions of the state-space as the sample density increases, the euclidean ball approach imposes more Lipschitz constraints maintaining the extent of continuity. We found it more convenient to work with the latter approach.

5.2.4 Out-Of-Sample

In the Out-Of-Sample settings, we allow the approximate Bellman operator to be defined over all state-action pairs. This implies that during policy execution, the approximate Bellman operator would have to be computed for every new state. Although this assumption restricts the applicability of this approach to small discrete action spaces for real-time applications, it poses significant benefits over the in-sample approaches. In most small domains, this method outperforms the other variations. An added advantage is that there is no need to impose explicit

[1] to avoid conflict with the k nearest neighbours used to approximate the Bellman operator

Lipschitz constraints. The continuity is incorporated into the approximate Bellman operator as in equation 3.1 reducing the computational complexity significantly.

Thus, given a reasonable coverage of the state-action space, it is beneficial to use the out-of-sample setting as long as the policy execution remains efficient.

5.2.5 More variations

The out-of-sample and in-sample settings are two extremes of a smooth transition. An intermediate case is when the Bellman operator is defined on unseen actions for seen states. This would retain the ability to deal with continuous action spaces but would result in a computationally more complex operator $T_{\mathbb{M}}$.

5.3 Conclusion

In this chapter, we uncovered some benefits of using the framework of induced MDPs to solve Lipschitz continuous value functions. Not only were we able to design efficient and easy-to-implement algorithms, thus lowering the conceptual and resource barriers for solving this problem, but we were also able to impose our bias in a far more controlled manner than was earlier possible. The one issue still pending to be resolved is the arbitrary choice of the Lipschitz constant L. In the next chapter, we introduce a gradient based algorithm to overcome this problem.

CHAPTER 6

Homotopy

The Lipschitz constant L which controls the degree of continuity enforced in the induced MDP $\mathbb{M} =< \mathcal{C}, L >$ is a continuous parameter. As we change the value of L, the optimal value function will change continuously, but the optimal policy changes at certain values of L and stays optimal for a range [1]. Here we present a homotopy based algorithm that helps us move across optimal policies for successive values of L. Although our initial aim was for using this algorithm as an alternate technique for computing the optimal value function similar to LARS [Efron *et al.*, 2004] (as suggested in Pazis and Parr, 2011), in our experiments, we noticed a very high density of solutions w.r.t L, especially in 'interesting' regions of L. Hence, it proved to be too expensive to jump through successive solutions due to the vast number of solutions. We discuss these findings in more detail in section 7.3. We present this approach nonetheless, as it has useful ideas that can come in handy for future applications.

For the discussion that follows, we return to the linear programming view of solving an induced MDPs. The terminology of 'slacks' and 'tight constraints' follow from this view.

1. In section 6.1, we evaluate the gradient of the optimal value function w.r.t the continuous parameter L.
2. In section 6.2, we use gradient value function to find the smallest increment in L for which the optimal policy changes. This gives us a basic homotopy-based algorithm
3. Section 6.3 talks about an alternate algorithm for homotopy that can be more efficient.

[1] The structure of valid policies ensures that there are only a finite number of policies possible for a finite sample set.

6.1 The Gradient Markov Chain

We make the crucial observation that for the continuous interval of L where the optimal policy remains fixed, the value function can be represented as a system of linear equations according to the tight constraints of the underlying LP (the maximizing expressions in $T_{\mathbb{M}}$. Differentiating w.r.t L, yields another system of linear equations. As we show, this system of equations represents the value function of a synthetic Markov chain which we call the *gradient Markov chain*.

We now derive the changes in value function denoted by G for a given valid policy π and L. π defines a set of inequalities which will continue to be tight until the 'next' L at which point the policy will change. We use the compressed characterization $(\Omega, \overline{d})$ of a valid policy from section 4.3.

$$V_L^*(s) = \begin{cases} R(s, \mathcal{A}(s)) + \gamma \sum_{s' \in \tilde{S}} P(s'|s, \mathcal{A}(s)) V_L^*(s') & s \in S_B \\ V_L^*(\Omega(s)) - L\overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow (-\frac{dV_L^*}{dL})(s) = \begin{cases} \gamma \sum_{s' \in \tilde{S}} P(s'|s, \mathcal{A}(s))(-\frac{dV_L^*}{dL})(s') & s \in S_B \\ (-\frac{dV_L^*}{dL})(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma[\sum_{s' \in S_B} P(s'|s, \mathcal{A}(s)) G(s') + \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s)) G(s')] & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma \sum_{s' \in S_B} P(s'|s, \mathcal{A}(s)) G(s') + \gamma \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s))[G(\Omega(s')) + \overline{d}(s', \Omega(s'))] & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma \sum_{s' \in S_B} P(s'|s, \mathcal{A}(s)) G(s') + \gamma \sum_{s' \in S_B} \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s))[G(s'')) + \overline{d}(s'', s')] & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma \sum_{s' \in S_B} \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s)) \overline{d}(s', s'') \\ + \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) + \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s))] G(s') & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma \sum_{s' \in S_B} \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s)) \overline{d}(s', s'') \\ + \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) + \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s))] G(s') & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

$$\Rightarrow G(s) = \begin{cases} \gamma \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s)) \overline{d}(s', \Omega(s')) + \gamma \sum_{s' \in S_B} [P(s'|s, \mathcal{A}(s)) \\ + \sum_{s'': \Omega(s'') = s'} P(s''|s, \mathcal{A}(s))] G(s') & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

Definition 28. *We can think of G as the fixed point value function for the following Markov*

Chain (MDP with a deterministic policy) defined by

$\mathcal{MC} =< S_B, \mathcal{A}, \Omega >$ -

- $S_{\mathcal{MC}} = S_B$
- $P_{\mathcal{MC}}(s'|s) = P(s'|s, \mathcal{A}(s)) + \sum_{s'':\Omega(s'')=s'} P(s''|s, \mathcal{A}(s))$
- $R_{\mathcal{MC}}(s) = \gamma \sum_{s' \in S_L} P(s'|s, \mathcal{A}(s))\overline{d}(s', \Omega(s'))$

6.2 Successive optimal policies

Definition 29. $\forall s \in S^*,\ \forall a \in Act(s)$,

1. *The bellman slacks are defined as* $V^*(s) - R(s,a) - \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')$. *For* $\forall s \in S,\ s' \in Lip(s)$
2. *The lipschitz slacks are defined as* $V^*(s) - V^*(s') + L\overline{d}(s, s')$.

$\mathcal{I}$ *represents the set of all slacks.*

Clearly, at the fixed point, the slacks are all non-negative (from the definition of $T_{\mathbb{M}}$. We can now use the gradient vector to find the rate of change of slacks w.r.t L and thereby determine the 'next' policy and the value of L for which the this policy is optimal.

Definition 30. $\forall I \in \mathcal{I}$, G_L^I *is the rate of change of slack w.r.t L defined as* G_L^I *given G.* G_L^I *can be of two fundamental types depending on the type of inequality*

- *For a bellman slack corresponding to state* s_i *and an action* a_k, $G_L^I = G(s_i) - \gamma \sum_{s_j \in S} P(s_j|s_i, a_k)G(s_j)$.
- *For a lipschitz slack for state* s_i *being constrained by* s_j, *the rate of change of slack is* $G_L^I = G(s_i) - G(s_j) - \overline{d}(s_i, s_j)$

Definition 31. $G_L^{\mathcal{I}}$*is a function mapping* $I \in \mathcal{I}$ *to* G_L^I

We can now determine when the next change will occur, but also the type of change in optimal policy. $L_{next} = L - \min_{I \in \mathcal{I}} \frac{slack(I)}{G_L^{\mathcal{I}}(I)}$. The argmin determines the the new entering slack and thereby determines the next optimal policy.

6.3 An efficient homotopy based method

From our experiments, we observed that the solution space is extremely dense especially for lower values of L. For any non-trivial range of L, even for a few thousand samples, the optimal policy changes thousands of times.

The bottleneck of the homotopy algorithm is evaluating the gradient for every slack $G_L^{\mathcal{I}}$ (especially for global Lipschitz continuity where the number of slacks grow quadratically with the number of samples). If we do have a thousand solutions in L=[1,2], say, we can prune out slacks we need not care about since values should not change all that much. The number of slacks we prone out has a direct bearing on the number Here, we propose a technique which takes advantage of this.

Given all the slacks $\mathcal{I}$ and the value function at L_0 and the jump intended - δ_L, it is possible to compute an upper bound on the gradient vector which gives us an idea of how extreme things can go in the δ_L range. We can use this bound to prune out slacks that certainly will not go to zero in the solutions arising in $[L_0, L_1 = L_0 + \delta_L]$.

6.3.1 Upper Bounding the Gradient

Recall the gradient value function,

$$G(s) = \begin{cases} \gamma \sum_{s' \in S_L} P(s'|s, a^*)\overline{d}(s', \Omega(s')) + \gamma \sum_{s' \in S_B} [P(s'|s, a^*) + \sum_{s'':\Omega(s'')=s'} P(s''|s, a^*)]G(s') & s \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & s \in S_L \end{cases}$$

where $\Omega(s)$ is the Bellman state s is attached to, S_L are the Lipschitz states and S_B are the Bellman states.

We now derive a technique to upper bound this value function. As we will see, the computational complexity of this method depends upon the form of the approximate Bellman operator. Here, we present this method tailored to the case where the the approximation is carried out as in section 3.1. There are two reasons for doing so. Firstly, this approximation is fairly general and useful. Secondly, it illustrates potential problems in using this method for some approximations.

Consider a sample (x, a, r, x'). For the state x, x' is the 'next state'.

Definition 32. *For any state x, $N(x)$ are the k-nearest neighbours of x (including x itself). In general, this is the set of all states whose values directly contribute to the Bellman approximation for state-action pair (x, a).*

Definition 33. *L(x) = $\{s'|s \in N(x) \wedge s' \in S_L\}$ is the set of all dependencies that are in S_L for a given policy.*

Definition 34. *B(x) = $\{s'|s \in N(x) \wedge s' \in S_B\} \cup \{\Omega(s')|s' \in L(x)\}$ is the set of all Bellman states for a policy that contribute to the Bellman approximation for the state-action pair (x, a).*

$$G(x) = \begin{cases} \frac{1}{k} \sum\limits_{s \in N(x)} \overline{d}(s,x) + \frac{\gamma}{k}[\sum\limits_{s' \in L(x)} \overline{d}(s', \Omega(s')) + \sum\limits_{s' \in B(x)} G(s')] & x \in S_B \\ G(\Omega(s)) + \overline{d}(s, \Omega(s)) & x \in S_L \end{cases}$$

Let G^* be the maximum gradient value for any Bellman state in all of the solutions that between L_0 and L_1.

$$\forall x \in S_B \;\; G(x) \leq \frac{1}{k} \sum_{s \in N(x)} \overline{d}(s,x) + \frac{\gamma}{k} \sum_{s' \in L(x)} \overline{d}(s', \Omega(s')) + \gamma G^*$$

$$\Rightarrow G^* \leq \gamma G^* + \max_x \frac{1}{k} \sum_{s \in N(x)} \overline{d}(s,x) + \frac{\gamma}{k} \sum_{s' \in L(x)} \overline{d}(s', \Omega(s'))$$

$$\Rightarrow G^* \leq \frac{\max\limits_x \frac{1}{k} \sum\limits_{s \in N(x)} \overline{d}(s,x) + \frac{\gamma}{k} \sum\limits_{s' \in N(x)} \overline{d}(s', \Omega(s'))}{(1-\gamma)} \qquad (1)$$

Note that in the last 2 steps, we have replaced $L(x)$ with $N(x)$ since we want an upper bound and although $L(x)$ changes in $[L_0, L_1]$, $N(x)$ doesn't.

Although it seems as if the problem is solved, there is one issue - as homotopy progresses in $[L_0, L_1]$, $\Omega(s)$ changes. Now, since we don't know $\overline{d}(s', \Omega(s'))$, we can use the maximum distance assuming that a Lipschitz state can attach to any Bellman state to upper bound. However, we can do better. Here is where the effect of δ_L comes in. For reasonable ranges, it seems likely that we can limit the possible $\Omega(s)$ to a subset of all states. Note that if we find that the candidate set is a null set, we have essentially solved the previous approximation.

6.3.2 The chicken and the egg

Let s be some state and V_0 be the optimal value function at L_0. We consider the lipschitz slacks at L_0 for the state s and eliminate those slacks that cannot go to 0. Obviously, this assumes that we know G^*. The lower G^*, the more slacks we can eliminate and the more slacks we can eliminate, the tighter the bound on G^* can be. We first assume that we have some G^* and derive the condition for eliminating a Lipschitz slack.

Recall that the rate of change of a Lipschitz slack is $G(s) - G(s') + \overline{d}(s, s') \leq G^* + \overline{d}(s, s')$ *throughout* $[L_0, L_1]$. Hence, we can remove all slacks where $\frac{S}{G^* + \overline{d}(s,s')} \geq \delta_L$, S being the current slack.

6.3.3 The algorithm

We first fix the δ_L for which we want to prune presenting a computational trade-off. On the one hand we want a large δ_L so that we can work for longer with the pruned slacks and on the other hand, the larger δ_L is, the fewer slacks get pruned.

Given δ_L, V_0 and S at L_0, For each state, initialize the pruned slack set as all the slacks.

1. Given pruned sets $P(s)$ for each state and these sets correspond candidate $\Omega(s)$.

$$G^* = \frac{\max_x \frac{1}{k} \sum_{s \in N(x)} \overline{d}(s, x) + \frac{\gamma}{k} \sum_{s' \in N(x)} \max_{s'' \in P(s')} (\overline{d}(s', s''))}{(1 - \gamma)}$$

2. Prune the sets according to G^*

Repeat until the pruned sets do not decrease or it gets small enough so that we are happy.

Although this method is expensive, it's cost will be amortized through all the iterations in δ_L.

Note: The effectiveness of the optimized homotopy method depends upon $N(x)$. In the ex-

ample above, it is evident that a larger k would decrease the effectiveness of this algorithm. Bellman approximations that average over a lot of states, in general, would result in large neighbourhood sets ($N(x)$) and would result in a G^* that is not very useful.

6.4 Conclusion

By introducing the concept of homotopy, we are no longer tied down to imposing a single Lipchitz constant L. The methods developed in this chapter allow us to seamlessly navigate through the policies arising from varying degrees of Lipschitz continuity.

CHAPTER 7

Experiments

The framework of induced MDPs introduces a variety of options in terms of algorithms and the degree of continuity imposed. We illustrate the importance of this framework with the help of series of experiments.

1. We compare the three algorithms described in this paper in terms of run time and show that the policy iteration approach, in addition to being easy to implement, is vastly superior to NP-ALP and value iteration by this metric. Further, we discuss the how each of these algorithms scales with the number of samples.
2. The degree of Lipschitz continuity imposed allows us to control the bias we impose on the problem. We compare the variations discussed in chapter 5.2 when applied to a real-world ball-balancing task.
3. Finally, we discuss homotopy and show that even for simple domains with a small sample set the solution density is extremely large making infeasible to use homotopy as a substitute for policy iteration.

Apart from providing empirical evidence on the advantages of imposing Lipschtiz continuity, this section also highlights the problems that need to be addressed in future work.

7.1 Algorithmic Improvements

We demonstrate the superiority of the policy iteration algorithm on the standard inverted pendulum domain my solving an in-sample-global MDP 5.2 and show significant improvements over NP-ALP. The trends seen in this section were observed for the physical 1D ball balancing task as well.

We replicated the inverted pendulum problem from Pazis and Parr [2011]. This involves balancing a pendulum in an upright position by applying force to the cart upon which it is mounted. The state space is continuous and 2-dimensional, being comprised of the vertical angle θ and angular velocity $\dot{\theta}$ of the pendulum.The action space is also continuous and consists of force $u \in [-50N, 50N]$. Our goal is to illustrate the benefits that policy iteration has over the other methods while dealing with large sample sets.

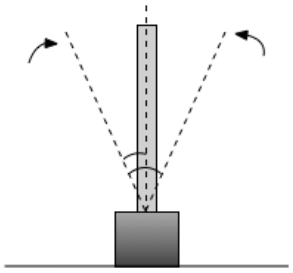


Figure 7.1: The inverted pendulum domain

Hence, we do not employ any dimensionality reduction techniques like PCA, which would simplify the problem significantly and obscure the effect of scaling. Therefore this experiment should not be seen as a benchmark for the sample complexity of NP-ALP.

We treated the problem as a regulation task with a reward of $1 - (u/50)^2$ given as long as $|\theta| \leq 2/\pi$ and 0 when $|\theta| > 2/\pi$, thus terminating the episode. A discount factor of 0.98 and control interval of 100ms were used. The two norm of the difference between states was used for the distance function and the Lipschitz constant set to $L_{\tilde{V}} = 10$.

In traditional MDPs, policy iteration is notable for its quick convergence. Table 7.1 demonstrates empirically that this is the case for the policy iteration algorithm for the induced MDP as well.

Each algorithm was used to solve for policies based on sample sets of varied sizes obtained from simulated episodes of the inverted pendulum with a uniform random policy. Figure 7.2

lists the CPU time required for each algorithm to solve the problem on a Core i7-4770 (Haswell) machine. Value iteration and policy iteration were implemented in a fairly straightforward manner in Matlab, while NP-ALP was implemented in Scala with Gurobi as the LP solver.

TRAJECTORIES	SAMPLES	POL ITER	VALUE ITER
250	1976	21	936
500	4079	24	724
1000	7966	23	879
2000	16104	25	983
4000	32128	22	829

Table 7.1: Comparison of the number of iterations needed for policy iteration to converge as opposed to value iteration. The results are for varying sample set sizes on the Inverted Pendulum domain.

For large training sets, a naive implementation of NP-ALP is unworkable because the quadratic number of constraints bogs down the LP solver. We used a highly customized version of constraint generation which is significantly more efficient in terms of time and memory. Although a detailed description of the algorithm is beyond the scope of this paper, the idea behind this algorithm is to add the sample corresponding to the most violated Bellman constraint, as defined in Pazis and Parr [2011], to the LP. At every stage, there is an active set of samples that constrain the LP and adding a new sample involves adding not only the Bellman constraint associated with the sample, but also the extra Lipschitz constraints needed to maintain correctness. Effectively, rather than solving for the entire sample set at once, this method maintains a list of active samples which keeps growing until no bellman constraint is violated. At this point, the LP is solved.

Figure 7.2 clearly demonstrates the efficiency of policy iteration. We now discuss the approximate asymptotic scaling properties of the three algorithms for NP-ALP's induced MDP. These

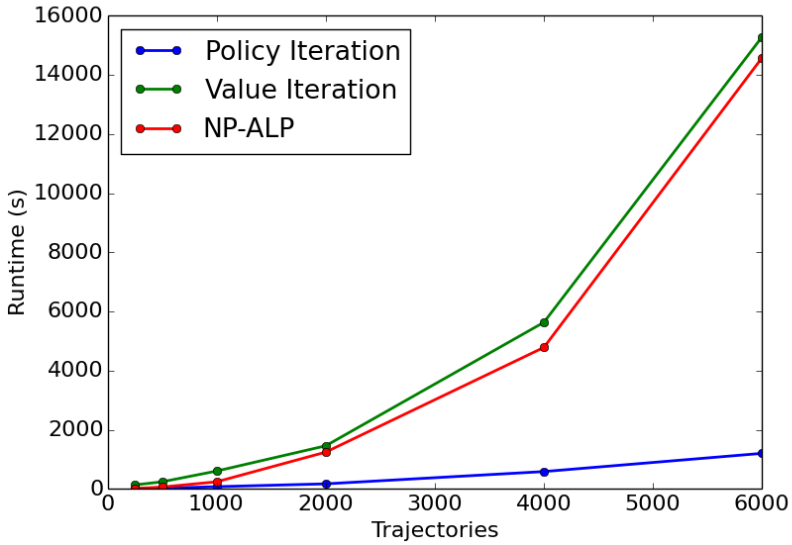


Figure 7.2: Runtime comparison of value iteration, policy iteration and NP-ALP. All timings are measured in seconds.

are useful as guidelines rather than proof of efficiency. Anecdotally, we note that even the highly optimized version of constraint generation will bog down for much larger training set sizes, magnifying the difference between policy iteration and NP-ALP.

With a constant number of iterations for policy evaluation, modified policy iteration scales as $\mathcal{O}(knm)$ where k is the number of nearest neighbours, n is the number of samples, and m is the average number of bellman constraints through the iterations. Value iteration also scales as $\mathcal{O}(knm)$ but the true run-time is highly dependant on the number of iterations needed for convergence. Different implementations of NP-ALP will have different scaling properties. While the naive implementation involves solving a linear program with $\mathcal{O}(n^2)$ constraints and $\mathcal{O}(n)$ variables, the customized version of constraint generation solves roughly $\mathcal{O}(m^*)$ linear programs, each with $\mathcal{O}(m^{*^2})$ constraints and $\mathcal{O}(km^*)$ variables each. Here, m^* is twice the number of bellman states in the optimal policy. Due to optimizations such as warm starts, this turns out to be much more efficient than even regular constraint generation.

7.2 1-D Ball Balancing

Figure 7.3: The ball balancing experimental setup for the reduced 1D case.

As an important step towards the final goal of controlling a quadcopter, we attempted to use the optimal Lipschitz continuous value function to control a robotic pan-tilt setup (Fig. 7.3) in order to balance a ball using weak sensors (a webcam).

The pan tilt was controlled using two Hi-Tec HS-422 servos and a SSC-32 controller board as shown in Fig 7.3, and a webcam (Logitech i920) operating at a resolution of 640×480 and at 30 frames per second to sense the position and velocity of the ball. Servo movements were restricted to ± 10 degrees about the mean centered position. The agent was penalized proportional to square of the deviation of the ball from the center of the board.

We detected a 3 frame lag in the vision sensing, and set the control frequency to 10 frames, including the previous action in the state space to preserve the decision process' Markovian nature. The problem was modelled into an MDP with a 6 dimensional state space and a 2 dimensional continuous action space.

We used this domain to compare the performance of the variations (section 5.2) of this frame-

work on this domain. We see that the out-sample variation performs better than the in-sample variations and is feasible due to the small action space (one dimensional).

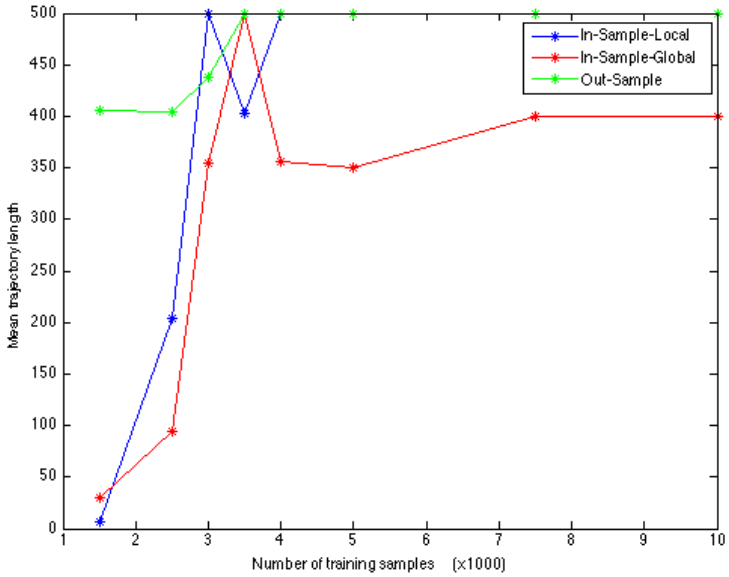


Figure 7.4: A comparative study of the in-sample-global, the in-sample-local and the out-sample variations.

The training data was collected from following a random policy. 20,000 samples were collected and used to generate random batches of different sizes. Each algorithm was then run on each of these batches and the policy obtained was tested over 20 trajectories and averaged. The number of timesteps (out of a maximum of 500) that the policy kept the ball balanced was taken as a measure of performance of the algorithm (Fig. 7.4).

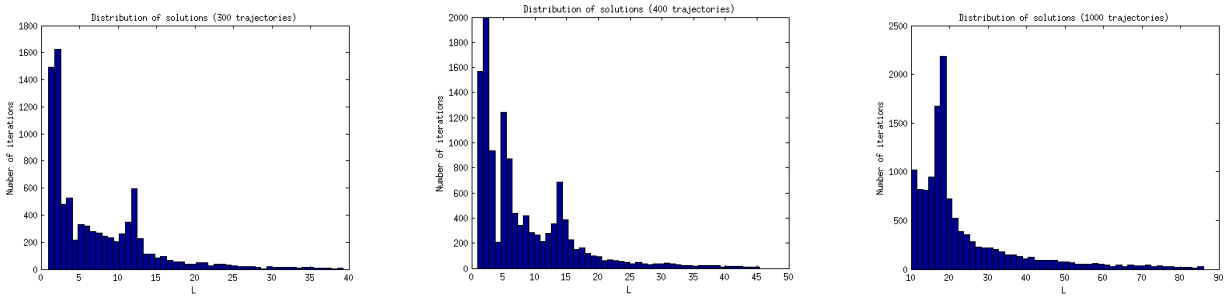
Overall, were able to learn a very neat policy for the reduced 1D case using only 6,000 samples in spite of the lag. [1] We built a simulator for the 2D ball balancing task incorporating friction and noise, but it was too far from reality and required only 20,000 samples to find a good policy. We are currently working on solving the physical 2D ball balancing task.

[1]The demo can be found here - https://www.dropbox.com/s/epu79xophbcsh9i/Ball%20Balance.mp4?dl=0

7.3 Homotopy

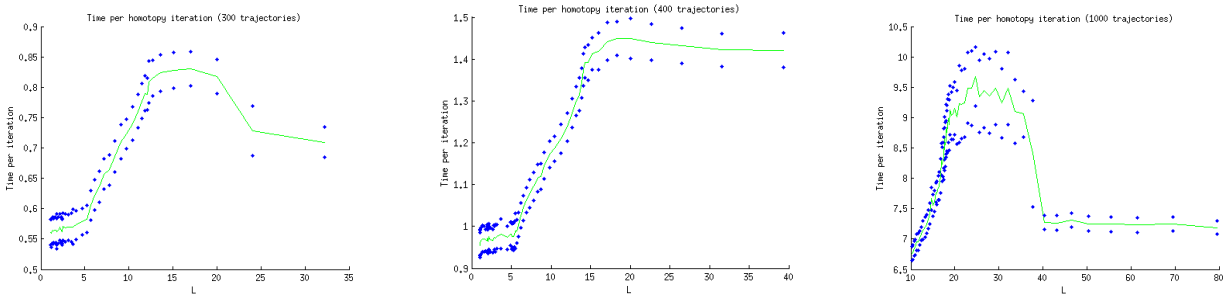
As the Lipschitz constant L changes, the induced MDP as well as the optimal policy change. The homotopy algorithm provides a unique opportunity to estimate the solution density (the rate at which the optimal policy changes w.r.t L).

On the standard Inverted Pendulum domain (section 7.1), we ran experiments to estimate the solution density in different regions of the parameter space (ranges of L) for 300, 400 and 1000 trajectories. As the sample set increases in size, the solution space becomes denser.



(a) (b) (c)

Figure 7.5: Number of solutions for (a) 300 trajectories and (b) 400 trajectories between $L = 1$ to $L = 50$ for (c) 1000 trajectories between $L = 10$ and $L = 100$ on the Inverted Pendulum domain.



(a) (b) (c)

Figure 7.6: Time taken per homotopy iteration for (a) 300 trajectories and (b) 400 trajectories between $L = 1$ to $L = 50$ for (c) 1000 trajectories between $L = 10$ and $L = 100$ on the Inverted Pendulum domain. The blue dots show the upper and lower confidence bounds while the green line is the mean time per iteration. One can see that the number of bins get sparser with increasing L, an observation consistent with 7.5

For figs. 7.5 and 7.6, the homotopy algorithm was run for a continuous range of the Lipschitz constant and statistics were binned in groups of 100 solutions each.

The inverted pendulum is a relatively small domain and for real-world problems, we need to be able to work with a huge number of samples. Although the time taken for each successive solution is very small making the the algorithm quite efficient, the solution space is too dense (Fig. 7.5)for a homotopy-based to be applied for finding the optimal policy from scratch.

Trajectories	Samples	L	Iterations	Time(hours)
300	2539	1 to 50	8784	1.6
400	3341	1 to 50	12463	4.2
1000	8392	10 to 100	13241	8.4

Table 7.2: Run time and solution density summaries for Fig. 7.5 and 7.6.

Instead, homotopy is well suited for a fine-grained parsing of the solution space. One can use a combination of homotopy and policy iteration in order to scan the entire state space and obtain the best possible Lipschitz continuous value function for the problem, given a set of samples. The only limitation for such an approach is that we need to ensure that the policies generated can be tested autonomously. This is a reasonable assumption and we plan to explore this option in the future.

CHAPTER 8

Future Directions

We believe that we have developed a comprehensive framework for learning Lipschitz continuous value functions. With the necessary theory and algorithms in place, we are able to find optimal value functions for large sample sets in continuous state-action spaces. While we have made significant improvements over the prior work of Pazis and Parr in this realm, we have also opened up several new issues that need to be answered:

1. When explicit Lipschitz continuity is beneficial
2. How one determines the optimal level of continuity needed
3. How this framework aids an expert policy

8.1 Large Domains

While it is clear that there are benefits of incorporating Lipschitz continuity in the value functions, in order to truly appreciate the effect of this bias, we need to move to larger domains where continuous actions are a must for controllability. The 2D ball balancing task is indeed one such domain, especially due to the added complexity from video lag. Another domain worth exploring is the helicopter domain[1].

[1]More about the domain here - https://sites.google.com/site/rlcompetition2014/domains/helicopter

8.2 Learning by demonstration

Incorporating domain knowledge is a vital component for building robotic reinforcement learning algorithms. One way to implement this is learning by demonstration with applications in social robotics. Imposing continuity is advantageous since good samples derived from an expert policy have a ripple effect thus spreading knowledge across states. In our experiments, we have been able to learn good policies for two standard domains (Mountain Car and Inverted Pendulum) in very few trajectories with limited demonstration. The next step is quantifying the effect of human demonstration and developing alternative techniques to add domain knowledge which are particularly suited for this framework.

8.3 Updating the Induced MDP online

An induced MDP defines a set of constraints. Until now, the algorithms developed are purely batch-RL algorithms in that they work on previously collected data. Let V_1^* be the optimal Lipschtiz continuous value function for batch $\mathbb{X}_1$ and let's assume that we now have access to some new samples, which after incorporating into $\mathbb{X}_1$, gives us a new batch $\mathbb{X}_2$. Then a key observation is that $V_2^* \succ V_1^*$ as long as the approximate Bellman operator is not re-evaluated for old samples. Thus, V_1^* can be used as the pessimistic initialization for the dynamic programming algorithms or as a warm start to the linear programming approach (chapter 5) solving for the new batch more quickly than if solved from scratch. This idea can be used to develop an online algorithm to solve induced MDPs.

REFERENCES

Barto, A. and **R. Crites** (1996). Improving elevator performance using reinforcement learning. *Advances in neural information processing systems*, **8**, 1017–1023.

Barto, A. G., *Reinforcement learning: An introduction*. MIT press, 1998.

Barto, A. G., **S. J. Bradtke**, and **S. P. Singh** (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**(1), 81–138.

Bellman, R. (1956). Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, **42**(10), 767.

Boyan, J. A. and **M. L. Littman** (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, 671–671.

de Farias, D. P. and **B. Van Roy** (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, **51**(6), 850–865.

De Farias, D. P. and **B. Van Roy** (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of operations research*, **29**(3), 462–478.

Efron, B., **T. Hastie**, **I. Johnstone**, **R. Tibshirani**, *et al.* (2004). Least angle regression. *The Annals of statistics*, **32**(2), 407–499.

Ernst, D., **P. Geurts**, and **L. Wehenkel**, Tree-based batch mode reinforcement learning. *In Journal of Machine Learning Research*. 2005.

Farahmand, A. M., **M. Ghavamzadeh**, **S. Mannor**, and **C. Szepesvári**, Regularized policy iteration. *In Advances in Neural Information Processing Systems*. 2009.

Feldbaum, A. (1960). Dual control theory. *Automation and Remote Control*, **21**(9), 874–1039.

Gosavi, A. (2004). A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, **55**(1), 5–29.

Hinderer, K. (2005). Lipschitz continuity of value functions in markovian decision processes. *Mathematical Methods of Operations Research*, **62**(1), 3–22.

Howard, R. A. (1960). Dynamic programming and markov processes..

Jaakkola, T., **M. I. Jordan**, and **S. P. Singh** (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, **6**(6), 1185–1201.

Kim, H., **M. I. Jordan**, **S. Sastry**, and **A. Y. Ng**, Autonomous helicopter flight via reinforcement learning. *In Advances in neural information processing systems*. 2003.

Kuvayev, D. and **R. S. Sutton** (1997). Model-based reinforcement learning. Technical report, Citeseer.

Lagoudakis, M. G. and **M. L. Littman**, Algorithm selection using reinforcement learning. *In ICML*. 2000.

Lagoudakis, M. G. and **R. Parr** (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, **4**, 1107–1149.

Li, L. (2009). *A unifying framework for computational reinforcement learning theory*. Ph.D. thesis, Rutgers, The State University of New Jersey.

Li, L., **V. Bulitko**, and **R. Greiner** (2007). Focus of attention in reinforcement learning. *J. UCS*, **13**(9), 1246–1269.

Mehta, D. and **D. Yamparala**, Policy gradient reinforcement learning for solving supply-chain management problems. *In Proceedings of the 6th IBM Collaborative Academia Research Exchange Conference (I-CARE) on I-CARE 2014*, I-CARE 2014. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-3037-4. URL http://doi.acm.org/10.1145/2662117.2662129.

Moore, A. W. and **C. G. Atkeson** (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, **13**(1), 103–130.

Munos, R. and **A. Moore** (2002). Variable resolution discretization in optimal control. *Machine learning*, **49**(2-3), 291–323.

Ng, A. Y., **A. Coates**, **M. Diel**, **V. Ganapathi**, **J. Schulte**, **B. Tse**, **E. Berger**, and **E. Liang**, Autonomous inverted helicopter flight via reinforcement learning. *In Experimental Robotics IX*. Springer, 2006, 363–372.

Pazis, J. and **R. Parr**, Non-Parametric Approximate Linear Programming for MDPs. *In AAAI*. 2011.

Pazis, J. and **R. Parr**, Pac optimal exploration in continuous space markov decision processes. *In AAAI*. 2013*a*.

Pazis, J. and **R. Parr**, Sample complexity and performance bounds for Non-Parametric Approximate Linear Programming. *In AAAI*. 2013*b*.

Petrik, M., **G. Taylor**, **R. Parr**, and **S. Zilberstein**, Feature selection using regularization in approximate linear programs for Markov decision processes. *In Proceedings of the 27th International Conference on Machine Learning*. 2010.

Ruvolo, P. L., **I. Fasel**, and **J. R. Movellan**, Optimization on a budget: A reinforcement learning approach. *In Advances in Neural Information Processing Systems*. 2009.

Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, **44**(1.2), 206–226.

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, **52**, 55–66.

Silver, D., **R. S. Sutton**, and **M. Müller**, Reinforcement learning of local shape in the game of go. *In IJCAI*, volume 7. 2007.

Singh, S. and **D. Bertsekas** (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Advances in neural information processing systems*, 974–980.

Taylor, G. and **R. Parr**, Kernelized value function approximation for reinforcement learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, **38**(3), 58–68.

Vlassis, N. and **M. Toussaint**, Model-free reinforcement learning as mixture learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009.

Williams, J. D. (2007). *Partially observable Markov decision processes for spoken dialogue management*. Ph.D. thesis, University of Cambridge.

Zhang, W. and **T. G. Dietterich**, A reinforcement learning approach to job-shop scheduling. *In IJCAI*, volume 95. Citeseer, 1995.