Non-parametric Approximate Linear Programming for MDPs

by

Jason Pazis

Department of Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Vincent Conitzer

_____
Mauro Maggioni

_____
Silvia Ferrari

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in the Department of Department of Computer Science
in the Graduate School of Duke University
2012

Abstract

Non-parametric Approximate Linear Programming for MDPs

by

Jason Pazis

Department of Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Vincent Conitzer

_____
Mauro Maggioni

_____
Silvia Ferrari

An abstract of a thesis submitted in partial fulfillment of the requirements for
the degree of Master of Science in the
Department of Department of Computer Science
in the Graduate School of Duke University
2012

Abstract

One of the most difficult tasks in value function based methods for learning in Markov Decision Processes is finding an approximation architecture that is expressive enough to capture the important structure in the value function, while at the same time not overfitting the training samples. This thesis presents a novel Non-Parametric approach to Approximate Linear Programming (NP-ALP), which requires nothing more than a smoothness assumption on the value function. NP-ALP can make use of real-world, noisy sampled transitions rather than requiring samples from the full Bellman equation, while providing the first known max-norm, finite sample performance guarantees for ALP under mild assumptions. Additionally NP-ALP is amenable to problems with large (multidimensional) or even infinite (continuous) action spaces, and does not require a model to select actions using the resulting approximate solution.

Contents

List of Figures

List of Abbreviations and Symbols

Symbols

π	Policy
γ	Discount factor $\in [0, 1)$.
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
$\mathcal{P}$	Transition model
$\mathcal{R}$	Reward function
$\mathcal{D}$	Initial state distribution

Abbreviations

RL	Reinforcement Learning
MDP	Markov Decision Process
ALP	Approximate Linear Programming
FQI	Fitted-Q iteration
PCA	Principal Component Analysis

Acknowledgements

Reflecting on the past few years, the question I find myself asking is how did I get here? The answer is not alone. While the decision to pursue graduate studies ultimately rests upon the individual, we are all influenced by our interactions with other people, often to a larger extent than we realize.

First of all, I would like to thank my former advisor Michail G. Lagoudakis. It is not very often that an advisor has such a profound effect on a student's career. Michail was the one who introduced me to reinforcement learning, believed in me to pursue my own ideas even as an undergrad, and motivated me to continue my studies at Duke. If it wasn't for Michail, if I was in graduate school at all, I would probably be studying something completely different.

Most students consider themselves lucky if they find *one* good advisor. During my studies, I've been fortunate enough to have two excellent advisors. I would like to thank my current advisor Ronald Parr, who had some difficult shoes to fill, yet has been able to meet and exceed my expectations. Ron has always managed to find a good balance between micro-management and being overly hands off, giving me the academic freedom to pursue my own ideas instead of pursuing his own agenda, while providing enough feedback, coupled with healthy doses of humor, for my research to be a collaboration rather than a one man project.

Apart from my advisors with whom I've had the most frequent interaction, I have to thank many other faculty members, who have helped me not only directly

with feedback on my ideas, but also indirectly by teaching me principles on how to think about and approach problems and research in general. Two members of my committee, Vince Conitzer, and Mauro Maggioni deserve special mention, as many of the ideas presented in this thesis were developed while I was attending their classes.

I would also like to thank all of my friends. One of the many ways that graduate school differs from a regular job is that surrounding yourself with friends is not just an advantage; its a requirement. Interacting with friends at various stages of their graduate studies, some further along, some just starting and some in the same stage, has been invaluable both in making decisions about my own future, and in making the past few years an enjoyable experience rather than a chore.

Last but not least, I would like to thank a number of organizations that supported me through my studies. In particular I would like to thank NSF for grant IIS-0713435, as well as ICML and AAAI. While sometimes as students we tend to forget about who's paying the bills, much of this research would not have been possible without their support.

1

Introduction

Learning from experience is one of the central abilities commonly associated with intelligence. Humans consider themselves intelligent in large part because of their ability to learn, and tend to treat animal species who are also proficient at learning preferentially. Recently, despite much skepticism, machines have joined living organisms in their ability to learn. In fact, machines today are much better than humans in learning certain types of tasks. Nevertheless, there are classes of problems, many of which also difficult for humans, where machines still have a long way to go.

This thesis deals with a particular class of problems called sequential decision problems, commonly modeled as a Markov Decision Process (MDP), and tackled by algorithms from the field of Reinforcement Learning (RL), a subfield of Machine Learning. Sequential decision problems differ from simpler "one-shot" problems in that there is some notion of a state, and actions influence that state. To make the distinction clearer, look at examples from both sequential and non-sequential problems. A classical "one-shot" problem would be a spam filter. Classifying a particular email as spam or not, does not alter the probability that the next email received will be spam. In contrast, in a patient treatment problem, administering a

certain drug at one point in the process may have a strong impact on the patient's status at the next step.

What makes sequential decision problems both interesting and challenging is that we usually don't have examples of "good" or "bad" behavior. Instead what we'll have, or what we may be able to collect, are samples from interaction with the process which will give us some idea of what the effects of different actions from various states are. Using this limited information, our task is to not only try to find the action that looks best at this time-step, but to try to balance long and short term goals.

1.1 Contribution

This thesis introduces Non-Parametric Approximate Linear Programming (NP-ALP), a batch mode reinforcement learning algorithm based on linear programming.

Linear programming is one of the standard ways to solve for the optimal value function of a Markov Decision Process. While its approximate, feature based version, Approximate Linear Programming (ALP), has been known for quite a while, until recently it had not received as much attention as approximate value and policy iteration methods. This can be attributed to a number of apparent drawbacks, namely poor resulting policy performance when compared to other methods, poor scaling properties, dependence on noise-free samples, no straightforward way to go from the resulting value function to a policy without a model and only $l1$-norm bounds. A recent surge of papers has tried to address some of these problems. One common theme among most of these papers is the assumption that the value function exhibits some type of smoothness.

Instead of using smoothness as an indirect way to justify the soundness of the algorithms, this work takes a very different approach, which relies on a smoothness assumption on the value function (not necessarily in the ambient space). NP-ALP

offers a number of important advantages over its feature based counterparts:

- The most obvious advantage is that because NP-ALP is non-parametric, there is no need to define features or perform costly feature selection.
- NP-ALP is amenable to problems with large (multidimensional) or even infinite (continuous) state and action spaces.
- NP-ALP offers significantly stronger and easier to compute performance guarantees than feature based ALP, the first (to the best of our knowledge) max-norm performance guarantees for any ALP algorithm.
- NP-ALP offers finite sample performance guarantees under mild assumptions, even in the case where noisy, real world samples are used instead of the full Bellman equation.

1.2 Thesis outline

Chapter 2 provides an introduction to Markov Decision processes, Reinforcement Learning, Approximate linear programming and other concepts used throughout the thesis.

Chapter 3 introduces the Non-Parametric Approximate Linear Programming approach, explains some of its most important properties, and concludes by proving max-norm performance guarantees and finite sample convergence.

Chapter 4 provides an overview of related work in the areas of Approximate Linear Programming and non-parametric RL methods.

Chapter 5 provides experimental results on three popular reinforcement learning domains. The car on the hill problem, the bicycle balancing problem and a highly noisy version of the Inverted Pendulum problem.

Chapter 6 concludes this thesis by summarizing the strengths and weaknesses of NP-ALP and gives a number of guiding directions for future work.

2

Background

2.1 Agents and environments

An agent can be anything that has the ability to perceive some aspect(s) of its environment and act. The environment is what our agent perceives as the "outside" world. It can be the real world, a room, a simulated labyrinth, or even an actuator. Considering an actuator as part of the environment, rather than part of the agent, may at first seem unnatural. However, in the cases that we are interested in, the agent will usually have no prior knowledge about the results of its own actions. This is very similar to the behavior of a newborn baby, who initially knows nothing about controlling his/her own arms and legs. Thus, our agents will treat everything as part of the environment, even their own actuators, and use their observations to learn how to interact with it in a beneficial way.

Environments are in most cases stochastic. What this means for our agent is that even if everything in the environment is the same between two repetitions of an experiment (the agent is in the same state), an action may have different outcomes. The difference between outcomes can be sharp as in the (discrete) case of a coin toss

that may produce heads or tails, or it can be more subtle and fine-grained, as in the (continuous) case of a motor that rotates within a certain speed/torque range when electric current is applied to its terminals.

2.2 Markov Decision Processes (MDP)

A Markov Decision Process (for an overview see Puterman (1994)), is a discrete-time mathematical decision-making modeling framework, that is particularly useful when the outcome of a process is a function of the agent's actions perturbed by external influences (such as noise). MDPs have found extensive use in areas such as economics, control, manufacturing, and reinforcement learning.

An MDP is defined as a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{D})$, where:

- $\mathcal{S}$ is the state space of the process. It can be finite set, or it can be infinite as is the case when there is some state variable that can take values in a continuous range. The current state s is assumed to be a complete description of the state of the environment at the current timestep.

- $\mathcal{A}$ is the action space of the process. Just like the state space, it can be finite or infinite. The set of actions consist of the possible choices an agent has at each timestep.

- $\mathcal{P}$ is a Markovian transition model, where $P(s'|s, a)$ denotes the probability of a transition to state s' when taking action a in state s. The Markov property, implies that the probability of making a transition to state s' when taking action a in state s, depends only on the current s and a and not on the history of the process.

- $\mathcal{R}$ is the reward function (scalar real-valued) of the process. It is Markovian as well, and $\mathcal{R}(s, a, s')$ represents the expected immediate reward for any transi-

tion from s to s' under action a at any timestep. The expected reward for a state-action pair (s, a), is defined as:

$$\mathcal{R}(s,a) = \sum_{s' \in S} \mathcal{P}(s'|s,a)\mathcal{R}(s,a,s') \quad .$$

- $\gamma \in [0, 1)$ is the discount factor. It is a way to express the fact that we care more about rewards received now than in the distant future. The reward we receive at the current timestep is weighted by 1, while future rewards are exponentially discounted by γ^t. In the extreme case where $\gamma = 0$, the problem degenerates to picking the action that will yield the largest immediate reward (supervised learning). As γ gets closer to 1, we may sacrifice short term benefits to achieve higher rewards later.

- $\mathcal{D}$ is the initial state distribution. It describes the probability each state in $\mathcal{S}$ has to be the initial state. In some problems most states have a zero probability, while few states (possibly only one) are candidates for being an initial state.

2.3 Policies and value functions

A policy π is a mapping from states to actions. It defines the response (which may be deterministic or stochastic) of an agent in the environment for any state it encounters and it is sufficient to completely determine its behavior. In that sense $\pi(s)$ is the action chosen in state s by the agent following policy π.

An optimal policy π^*, is a policy that yields the highest expected utility in the long run. That is, it maximizes the expected total discounted reward under all conditions (over the entire state space). For every MDP there is at least one such policy although it may not be unique (multiple policies can be optimal; hence yielding equal expected total discounted reward through different actions).

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:

$$V^{\pi}(s) = E_{a_t \sim \pi; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, a_t, s_{t+1}\big) \Big| s_0 = s\right].$$

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, action a is taken at the first step, and all decisions thereafter are made according to policy π:

$$Q^{\pi}(s,a) = E_{a_t \sim \pi; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, a_t, s_{t+1}\big) \Big| s_0 = s, a_0 = a\right].$$

The goal of the decision maker is to find an optimal policy π^* for choosing actions, which maximizes the expected, total, discounted reward for states drawn from $\mathcal{D}$:

$$\pi^* = \arg\max_{\pi} E_{s \sim \mathcal{D}}\left[V^{\pi}(s)\right] = \arg\max_{\pi} E_{s \sim \mathcal{D}}\left[Q^{\pi}\big(s, \pi(s)\big)\right].$$

The optimal state and state-action value functions V^* and Q^*, can be defined recursively via the Bellman optimality equations[1]:

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

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

For every MDP, there exists at least one optimal, deterministic policy. If the optimal value function V^* is known, an optimal policy can be extracted only if the full MDP model of the process is also known to allow for one-step look-aheads. On

[1] To avoid the use of excessive notation, we'll use sums instead of integrals throughout the thesis, even though our results hold for continuous state-action spaces under mild assumptions.

the other hand, if Q^* is known, a greedy policy, which simply selects actions that maximize Q^* in each state, is an optimal policy and can be extracted without the MDP model. Value iteration, policy iteration, and linear programming are well-known methods for deriving an optimal policy, a problem known as planning, from a (not too large) discrete MDP model.

2.4 Reinforcement Learning (RL)

In reinforcement learning (Kaelbling et al., 1996; Sutton and Barto, 1998), a learner interacts with a stochastic process modeled as an MDP. It is usually assumed that the agent knows nothing about how the environment works or what the results of its actions are (does not have access to the model P and reward function R of the underlying MDP). Additionally, in contrast to supervised learning, there is no teacher to provide samples of correct or bad behavior.

The goal is to learn an optimal policy using the experience collected through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, essentially sampling the transition model and the reward function of the process. Thus, experience comes in the form of (s, a, r, s') samples.

Two related problems fall within reinforcement learning: *prediction* and *control*. In prediction problems, the goal is to learn to predict the total reward for a given fixed policy, whereas in control the agent tries to maximize the total reward by finding a good policy. These two problems are often seen together, when a prediction algorithm evaluates a policy and a control algorithm subsequently tries to improve it.

The learning setting is what characterizes the problem as a reinforcement learning problem, rather than the algorithms used to attack it. This means that very diverse algorithms coming from different backgrounds can be used and that is indeed the case. Most of the approaches can be distinguished into Model-Based learning and

Model-Free learning.

In Model-Based learning, the agent uses its experience in order to learn a model of the process and then find a good decision policy through planning. Model-Free learning on the other hand, tries to learn a policy directly without the help of a model. Both approaches have their strengths and weaknesses (in terms of guarantees of convergence, speed of convergence, ability to plan ahead, and use of resources).

2.5 Value function approximation

In many real world applications, the number of state-action pairs is too large (or even infinite if the state or action spaces are continuous), rendering exact representation impractical. In addition to the fact that some spaces are too large to be represented exactly, we may not have samples for every state-action pair, or processing all the samples may exceed our computational resources. In such cases we are forced to use some form of function approximation.

Practically all known function approximation methods from supervised learning have been used with reinforcement learning, including neural networks, decision trees and forests, nearest neighbors, linear combinations of (possibly non-linear) basis functions and kernels.

These function approximation methods offer different tradeoffs regarding representational power, convergence when combined with particular learning algorithms, generalization ability, number of parameters to be learnt and human effort required in picking appropriate parameters, with no method having a clear advantage over all others in all situations.

2.6 Solving MDPs via linear programming

One way to solve for the optimal value function V^* is via linear programming, where every state $s \in \mathcal{S}$ is a variable and the objective is to minimize the sum of the states'

values under the constraint that the value of each state must be greater than or equal to all Q-values for that state. We'll call these constraints the Bellman constraints:

$$
\begin{aligned}
&\textit{minimize} \ \sum_{s} V^*(s) \\
&\textit{subject to}: \\
&(\forall s, a) V^*(s) \geqslant \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma P(s'|s,a) V^*(s')\right)
\end{aligned}
$$

Extracting the policy is fairly easy (at least conceptually), just by picking the action with a corresponding non-zero dual variable for the state in question (equivalently, picking the action which corresponds to the constraint that has no slack in the current state). Note that we can have a set of state-relevance weights $\rho(s)$ associated with every state in the optimization criterion; however, for the exact case every set of positive weights leads to the same V^*.

2.7 Approximate linear programming (ALP)

As mentioned in section 2.5, in many real world applications the number of states is too large (or even infinite if the state space is continuous), rendering exact representation (using one variable per state) impossible. In those cases, the typical approach is to approximate the value function via a linear combination of (possibly non-linear) basis functions or features. The variables in the approximate linear program are now the weights assigned to each basis function and the value of each state is computed as $\phi(s)^T w$, where $\phi(s)$ is the feature vector for that state, and w is the weight vector. The linear program becomes:

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

Using features dramatically reduces the number of variables in the program, but does not reduce the number of constraints. Since the number of constraints is larger than the number of variables in the exact linear program, we have to find a way to reduce the number of constraints by sampling or constraint generation. Making certain assumptions over our sampling distribution (de Farias and Van Roy, 2004), or if we incorporate regularization (Petrik et al., 2010), we can sample constraints and bound the probability that we will violate a non-sampled constraint, or bound the performance degradation that will occur as a result of missing constraints.

Unfortunately this approximate formulation does not allow for easy extraction of a policy from the dual. Not only is the number of dual variables large (the same as the number of samples) but it does not offer a straightforward way to generalize to unseen states. Choosing an action using an ALP solution typically requires a model to compute Q-values given the approximate value function returned by the linear program. Also note that the state-relevance weights now influence the solution, imposing a trade-off in the quality of the approximation across different states (de Farias and Van Roy, 2003).

An alternative way of expressing the ALP (consistent with the notation in Petrik et al. 2010) that also emphasizes the similarity to the exact LP is to express the problem as an optimization within a constrained family of value functions:

$$
\begin{aligned}
&\textit{minimize} \sum_s \rho(s)\tilde{V}(s) \\
&\textit{subject to}: \\
&(\forall s,a)\tilde{V}(s) \geqslant \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma\tilde{V}(s')\right) \\
&\tilde{V} \in \mathcal{M}
\end{aligned}
$$

This representation is more general than the typical ALP formulation in that it allows arbitrary constraints on $\tilde{V}$ via $\mathcal{M}$. For the standard ALP approach, $\mathcal{M} = span(\Phi)$,

but any specification of $\mathcal{M}$ that can be implemented through linear constraints can be seen as a form of ALP.

2.8 Bellman Backup Operators

There are two common definitions of the Bellman operator for a Q value function. The first is the *optimal* Bellman operator B defined as:

$$BQ(s,a) \quad = \quad \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a'} Q(s',a') \right)$$

The Bellman backup operator B^π can also be defined for a particular policy π as:

$$B^\pi Q(s,a) \quad = \quad \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma Q(s',\pi(s')) \right)$$

An important consequence of the way the Bellman operator is defined, is that for $i \to \infty$, $B^i Q(s,a) \to Q^*(s,a)$ and $\left(B^\pi\right)^i Q(s,a) \to Q^\pi(s,a)$.

2.9 Hoeffding's inequality

Hoeffding's inequality[2] (Hoeffding, 1963) (also frequently referred to as Hoeffding's bound) which we will use extensively through this work provides a bound on the probability that the sum of n independent random variables deviates from its mean by more than a threshold t.

Let $X_1, X_2, \ldots X_n$ be independent random variables with finite first and second moments, $a_i \leqslant X_i \leqslant b_i$ and $\bar{X}$ their empirical mean. Then:

$$Pr\left(|\bar{X} - E[\bar{X}] \geqslant t|\right) \quad \leqslant \quad 2e^{-\frac{2t^2 n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$$

[2] Note that many forms of Hoeffding's inequality exist. Here we explain the one most relevant to this work.

2.10 Limitations of existing value function based approaches

Many approximate value function based approaches exist, not only of the approximate linear programming variety (de Farias and Van Roy, 2003), but also in various forms of approximate policy iteration (Lagoudakis and Parr, 2003), approximate value iteration (Ernst et al., 2005), Q-learning (Watkins, 1989) and temporal difference learning (Sutton and Barto, 1998).

There are three main limitations with current value function based approaches. The first is that most algorithms require a significant amount of human effort. This is both to fine-tune the approximation architecture parameters such as the number, type and position of features for parametric architectures, as well as parameters of the algorithm itself such as the learning rate for online algorithms.

The second is that many algorithms offer no, or very weak convergence guarantees, especially for a finite number of samples. This also depends a lot on the approximation architecture used, as for instance approximate value iteration can be shown to converge for certain classes of tree-based approximators, but can diverge when combined with neural networks or even linear combinations of basis functions.

Finally, the majority of algorithms that provide some sort of performance guarantees, express those guarantees in quantities that are difficult to obtain, often as hard as solving the problem optimally in the first place.

Our goal with NP-ALP is to provide an algorithm which requires little human effort and provides strong theoretical guarantees expressed in quantities that can be easily estimated or bounded.

3

Non-parametric ALP

In this chapter we introduce Non-Parametric Approximate Linear Programming (NP-ALP), discuss some of its most important properties, and conclude by proving max-norm, finite sample performance guarantees.

3.1 Assumptions

The main assumption required by NP-ALP is that there exists some distance function d on the state-action space of the process, for which the value function is Hölder continuous. A Hölder continuous action-value function satisfies the following constraint for all (s, a) and (s', a') pairs:

$$\exists\ (Q^*, \alpha) : |Q^*(s, a) - Q^*(s', a')| \leqslant C_{Q^*} d(s, a, s', a')^\alpha$$

where:

$$d(s, a, s', a') = ||k(s, a) - k(s', a')||$$

and $k(s, a)$ is a mapping from state-action space to a normed vector space. We assume $\alpha \in (0, 1]$. For $\alpha < 1$ we require $d(s, a, s', a') \in [0, 1]$ while for Lipschitz continuous value functions ($\alpha = 1$), $d(s, a, s', a')$ can be greater than 1.

For simplicity, in the following, we assume that the distance between two states is minimized for the same action: $\forall\ (a, a'),\ d(s, s') = d(s, a, s', a) \leqslant d(s, a, s', a')$. Thus we have for a Hölder continuous value function:

$$|V^*(s) - V^*(s')| \leqslant C_{V^*} d(s, s')^\alpha,$$

and it is easy to see that $C_{V^*} \leqslant C_{Q^*}$.

The notation $\mathcal{M}_{C,\alpha}$ denotes the set of functions with Hölder constants C, α. For any C and α, $\tilde{V} \in \mathcal{M}_{C,\alpha}$ can be enforced via linear constraints, which we'll call smoothness constraints:

$$\left(\forall s, s' : d^\alpha(s, s') < \frac{Q_{\max} - Q_{\min}}{C_{\tilde{Q}}}\right) \tilde{V}(s) \quad \geqslant \quad \tilde{V}(s') - C_{\tilde{Q}} d(s, s')^\alpha, \qquad (3.1)$$

where $Q_{\max}$ and $Q_{\min}$ are defined as: $Q_{\max} = \frac{R_{\max}}{1-\gamma}$ and $Q_{\min} = \frac{R_{\min}}{1-\gamma}$.

3.2 Definition

Using $\tilde{S}$ to represent the set of state-action pairs for which the Bellman equation is enforced, the non-parametric approximate LP will be:

$$\begin{aligned}
&\textit{minimize} \sum_{s \in \tilde{S}} \tilde{V}(s) \\
&\textit{subject to}: \\
&(\forall (s, a) \in \tilde{S}) \tilde{V}(s) \geqslant \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \tilde{V}(s')\right) \\
&\tilde{V} \in \mathcal{M}_{C_{\tilde{V}}, \alpha}
\end{aligned}$$

When $\tilde{V} \in \mathcal{M}_{C_{\tilde{V}}, \alpha}$ is implemented as in equation 3.1, this provides a complete specification of NP-ALP.

If $C_{\tilde{Q}} = C_{Q^*}$ and all Bellman constraints are present in the LP, the smoothness constraints will have no effect on the solution and $\tilde{V} = V^*$.

In practice $C_{\tilde{Q}}$ will typically differ from C_{Q^*}, either by design (to avoid overfitting) or due to lack of sufficient knowledge of C_{Q^*}. Also, for realistic applications it will be impractical to have one constraint per state-action, either because we will not have samples for every state-action (unknown model), or because of computational limitations.

3.3 Key properties

3.3.1 Sparsity

Notice that the smoothness constraint on $\tilde{V}$ is defined over the entire state space, not just the states in $\tilde{S}$. However, it suffices to implement smoothness constraints only for states in $\tilde{S}$ or reachable in one step from a state in $\tilde{S}$, as smoothness constraints on other states will not influence the solution of the LP.

We will call all (primal) variables corresponding to state-action pairs for which the corresponding Bellman constraint holds with equality *basic* and the rest *non-basic*. Non-basic variables (and their corresponding constraints) can be discarded without changing the solution. This is useful both for sparsifying the solution to make evaluation significantly faster, and can be used to solve the linear program efficiently either by constraint generation, or by constructing a homotopy method.

Consider a state-action pair s, a corresponding to a non-basic variable. This implies $\tilde{V}(s) = \tilde{V}(s') - C_{\tilde{V}} d(s, s')^\alpha$ for some state[1] s'. When presented with state s'' to evaluate, we have:

$$\tilde{V}(s'') \geqslant \tilde{V}(s) - C_{\tilde{V}} d(s'', s)^\alpha \tag{3.2}$$

$$\tilde{V}(s'') \geqslant \tilde{V}(s') - C_{\tilde{V}} d(s'', s')^\alpha \tag{3.3}$$

Substituting $\tilde{V}(s) = \tilde{V}(s') - C_{\tilde{V}} d(s, s')^\alpha$ into 3.2:

$$\tilde{V}(s'') \geqslant \tilde{V}(s') - C_{\tilde{V}} (d(s'', s)^\alpha + d(s, s')^\alpha) \tag{3.4}$$

[1] In the case where $s = s'$ this means that some other action dominates action a for state s.

Since $d(s'', s')^\alpha \leqslant d(s'', s)^\alpha + d(s, s')^\alpha$ for $\alpha \in (0, 1]$ (remember that $d \in [0, 1]$ for $\alpha \neq 1$), constraint 3.2 does not influence the value of $\tilde{V}(s'')$.

Finally, adding states to the objective function that are not in $\tilde{S}$ or weighting the states in $\tilde{S}$ would not alter the LP solution; thus it suffices to set the objective function to be the sum over only the states in $\tilde{S}$.

3.3.2 The NP-ALP solution can be stored and used efficiently

Since NP-ALP does not explicitly produce a value function for states not in the sample set, one may wonder how the value of a new state can be inferred given an NP-ALP solution. It suffices to store the variables corresponding to state-action pairs for which the corresponding Bellman constraint holds with equality[2].

When presented with an unknown state t that was not directly constrained in the LP, an estimate of its value can be easily obtained by exploiting Hölder continuity. $\tilde{V}(t)$ will be greater than or equal to $\tilde{V}(s_i) - C_{\tilde{V}} d(s, t)^\alpha$ for all s_i in $\tilde{S}$. This way the value that state t would have been assigned by the LP if a smoothness constraint on t had been included, can be easily determined.

3.3.3 NP-ALP allows model-free continuous action selection

For some query state s, the Bellman constraint that bounds the value of this state also bounds the maximal Q-value for this state. This means that actions in $\tilde{S}$ can come from a continuous range and that the maximizing action for any state can be found efficiently[3], but it does limit actions selected at execution time to actions available for some nearby state in $\tilde{S}$.

After non-basic variables have been discarded, there is only one surviving (both

[2] Which in the worst case is equal to the number of samples, or significantly less in most realistic situations.

[3] For simplicity we assume that all actions are available in all states. When this is not the case we'd have to take the distance of the sampled actions to the closest available action at the query state into account.

primal and dual) variable per basic state. For any basic state s, $\tilde{V}(s)$ is bounded by a Bellman constraint from state-action pair s, a, so $\tilde{V}(s) = \tilde{Q}(s, a)$. If s bounds the value of a non-basic state t by[4] $\tilde{V}(t) \geqslant \tilde{V}(s) - C_{\tilde{V}} d(s, t)^{\alpha}$, it also bounds $\tilde{Q}(t, a)$. The predicted optimal action at t will therefore be the same as in s since the bounds from other states are lower, implying lower estimated Q-values.

The above has two important consequences. First, only actions present in the training set can ever be selected during policy execution, since the value estimation and action selection mechanisms are pessimistic. Second, action selection complexity is independent of the number of actions, allowing us to deal with spaces with infinite (continuous) or massive (multidimensional) action spaces. Sampling is of course important; however, this goes beyond the scope of this chapter.

3.3.4 The NP-ALP is always well defined

The Hölder continuity constraints ensure that the solution is always bounded, even when large parts of the state-action space have been poorly sampled. This is in contrast to parametric ALP, where a single missing constraint can, in the worst case, cause the LP to be unbounded.

3.4 Practical considerations

Some readers will have noticed that in a naive implementation, the number of constraints scales quadratically with the number of samples in the worst case (when $\frac{Q_{\max} - Q_{\min}}{C_{\tilde{Q}}}$ spans the entire space), which could cause LP-solvers to bog down. Fortunately the NP-ALP constraints have a number of favorable properties. All the Hölder constraints involve exactly two variables, resulting in a very sparse constraint matrix, a property that modern solvers can exploit. Additionally, for some distance

[4] For simplicity of exposition, we assume that $C_{Q_a} = C_V \ \forall a \in A$. The case where different actions have different Hölder constants extends naturally.

functions, such as the $l1$ or max-norm, most (depending on the dimensionality of the space) Hölder constraints can be pruned.

Even in the case of an "unfriendly" norm, we can use an iterative approach, progressively adding samples whose Bellman constraint is violated. Taking advantage of the fact that solutions tend to be very sparse, and that samples whose Bellman constraint is not tight will not influence the solution, very large problems can be solved without ever adding more than a tiny fraction of the total number of constraints. In our experiments, this technique proved to be far more effective than naive constraint generation.

Finally, for every sample either its Bellman constraint or exactly one of its Hölder constraints will be active, which means we can construct a homotopy method[5]. Starting from $C_{\tilde{V}} = 0$ only one Bellman constraint will be active and all other states will be bound by Hölder constraints to $\tilde{V} = \frac{R_{\max}}{1-\gamma}$. Progressively relaxing $C_{\tilde{V}}$, the entire space of solutions can be traversed.

3.5 Geodesic distances

Often specifying a good global distance function can be challenging, especially in domains with very large state and action spaces. As shown in this section, the task can be greatly simplified in domains where our samples lie on some manifold in the space.

3.5.1 Definition and construction

Consider the graph produced by connecting each sample to all its neighbors that are at most d_{max} apart. To define what d_{max} means we will need a distance function $d(s, s, a, a')$ satisfying the triangle inequality as above; however, this time it will be used only to define local distances. The shape of the graph will define global dis-

[5] At the moment we have not yet implemented such a method.

tances. For parts of the state-action space that are densely populated with samples, geodesic distances will approach straight line distances as the number of samples increases. On the other hand, parts of the space with large, inaccessible gaps will have no direct interaction.

Interestingly, if we are using a regular LP solver, we do not need to explicitly compute any distances other than the local ones. Simply omitting Hölder constraints for samples that are not directly connected in the graph, leads to the same value function as running all pairs shortest paths and using the result with the original LP formulation.

The fact that the above is true may not be immediately obvious, so let us examine the two linear programs. Both will have the same set of Bellman constraints, the first is the one will have all the Hölder constraints generated after running all pairs shortest paths, while the second one will have only the local Hölder constraints. Since the constraints on the first LP are a superset of the constraints on the second one, all we need to show is that all the extra constraints are redundant.

Consider the Hölder constraint from state s_m to state s_1 in the first linear program. If the shortest path in the graph between states s_1 and s_m is $s_1, s_2, \ldots, s_m$, the constraint will be:

$$V(s_1) \geqslant V(s_m) - C_{\tilde{V}} \left(d^{\alpha}(s_1, s_2) + \cdots + d^{\alpha}(s_{m-1}, s_m) \right). \tag{3.5}$$

In addition to 3.5, both LPs will have the following constraints:

$$\begin{aligned} V(s_1) &\geqslant V(s_2) - C_{\tilde{V}} d^{\alpha}(s_1, s_2) \\ V(s_2) &\geqslant V(s_3) - C_{\tilde{V}} d^{\alpha}(s_2, s_3) \\ &\ldots \\ V(s_{m-1}) &\geqslant V(s_m) - C_{\tilde{V}} d^{\alpha}(s_{m-1}, s_m) \end{aligned}$$

Starting from the last constraint and substituting all the way to the first, we can see

that 3.5 is redundant.

3.5.2 Geodesic versus ambient space distances

The first and most obvious benefit of using geodesic distances, is that for the same distance function, the Hölder constant may be much smaller, requiring much fewer samples to achieve the same precision. Alternatively, for the same constant, the space of representable value functions is potentially much larger, reducing errors caused by our inability to represent the true value function. These benefits stem from the fact that parts of the space with large, inaccessible gaps will have no direct interaction[6].

The second major advantage of using geodesic distances is that the new linear program contains only a small fraction of the original constraints, allowing the efficient solution of much larger programs, even with a standard LP solver.

The biggest disadvantage of using geodesic distances is that we now have to retain all samples in the final solution. This is because long range distances are implicitly defined by the samples[7]. Even if a sample s_b is dominated locally by s_a, it may provide the best bound for a query point that is too far to be constrained by s_a directly.

3.6 Error bounds

In the following, B is used to denote the Bellman operator.

Definition 1. *Let $\tilde{V}$ be the solution to the NP-ALP.*

Definition 2. *Let $\tilde{Q}$ be the Q value function implied by the constraints of the NP-ALP.*

[6] Consider for example a state space lying on a manifold that resembles a swiss roll, with the value function being $V_{\min}$ at the center of the roll and growing with the shape of the manifold to $V_{\max}$ at the edge. There may be sudden "jumps" in the value function in the ambient space -between folds-, even though it will be very smooth (by construction) in the manifold space.

[7] Of course this is not too different from using a manifold learning algorithm on the samples first and applying the original algorithm on the result.

Lemma 3. *(Second part of theorem 4.1 in Williams and Baird 1993). Let* $\epsilon = ||Q' - BQ'||_\infty$ *denote the Bellman error magnitude for* Q*, and* $V^* \leqslant V_Q$*. The return* V^π *from the greedy policy over* Q *satisfies:*

$$\forall s \in \mathcal{S},\ V^\pi(s) \geqslant V^*(s) - \frac{\epsilon}{1-\gamma}$$

Lemma 4. *Let* $\epsilon \geqslant 0$ *be a constant such that:* $\forall (s,a) \in (\mathcal{S}, \mathcal{A}), BQ(s,a) \leqslant Q(s,a) + \epsilon$*. Then:*

$$\forall (s,a) \in (\mathcal{S}, \mathcal{A}), Q^*(s,a) \leqslant Q(s,a) + \frac{\epsilon}{1-\gamma}$$

Proof. We will prove our claim by induction. All we need to prove is that $B^i Q_0(s,a) \leqslant Q_0(s,a) + \sum_{j=0}^{i-1} \gamma^j \epsilon$, and then take the limit as $i \to \infty$.

The base is given by hypothesis. Assuming that $B^i Q_0(s,a) \leqslant Q_0(s,a) + \sum_{j=0}^{i-1} \gamma^j \epsilon$, we'll prove that the inequality also holds for $i+1$:

$$\begin{aligned}
B^{i+1}Q_0(s,a) &= BB^iQ_0(s,a) \leqslant \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} B^iQ_0(s',a')\right) \\
&\leqslant \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma\left(\max_{a'} Q_0(s',a') + \sum_{j=0}^{i-1}\gamma^j\epsilon\right)\right) \\
&= \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_0(s',a')\right) + \gamma\sum_{j=0}^{i-1}\gamma^j\epsilon \\
&\leqslant Q_0(s,a) + \epsilon + \gamma\sum_{j=0}^{i-1}\gamma^j\epsilon \\
&= Q_0(s,a) + \sum_{j=0}^{i}\gamma^j\epsilon
\end{aligned}$$

If we now take the limit as $i \to \infty$ we have the original claim:

$$\lim_{i\to\infty} B^iQ_0(s,a) \leqslant Q_0(s,a) + \sum_{j=0}^{i-1}\gamma^j\epsilon \to Q^*(s,a) \leqslant Q_0(s,a) + \frac{\epsilon}{1-\gamma}$$

□

Lemma 5. *Let $V^* \leqslant V'$. Then:*

$$||V' - BV'||_\infty \leqslant ||V' - V^*||_\infty$$

Theorem 6. *Let $\epsilon_- \geqslant 0$ and $\epsilon_+ \geqslant 0$ be constants such that: $\forall (s,a) \in (\mathcal{S}, \mathcal{A}), -\epsilon_- \leqslant Q(s,a) - BQ(s,a) \leqslant \epsilon_+$. The return V^π from the greedy policy over Q satisfies:*

$$\forall s \in \mathcal{S},\ V^\pi(s) \geqslant V^*(s) - \frac{\epsilon_- + \epsilon_+}{1-\gamma}$$

Proof. We set $Q'(s,a) = Q(s,a) + \frac{\epsilon_-}{1-\gamma},\ \forall (s,a) \in (\mathcal{S}, \mathcal{A})$. It's easy to see that the performance achieved by the one step greedy policy π over Q and π' over Q' is the same: $V^\pi(x) = V^{\pi'}(x)$. From lemma 4 we have $\forall (s,a) \in (\mathcal{S}, \mathcal{A}), Q'(s,a) \geqslant Q^*(s,a)$.

$$\begin{aligned}
\forall (s,a) \in (\mathcal{S}, \mathcal{A}),\ BQ'(s,a) &= \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a'} \left(Q(s',a') + \frac{\epsilon_-}{1-\gamma} \right) \right) \\
&= BQ(s,a) + \gamma \frac{\epsilon_-}{1-\gamma} \\
\Rightarrow Q'(s,a) - BQ'(s,a) &= Q(s,a) - BQ(s,a) + \epsilon_- \\
&\leqslant \epsilon_- + \epsilon_+
\end{aligned}$$

From lemma 3 we have $\forall s \in \mathcal{S},\ V^\pi(s) \geqslant V^*(s) - \frac{\epsilon_- + \epsilon_+}{1-\gamma}$. □

Definition 7. *Let ϵ_s^- and ϵ_s^+ denote the maximum underestimation and overestimation Bellman error respectively, introduced by using sampled transitions instead of the exact transition probabilities, such that:*

$$\begin{aligned}
\forall (s,a) \in (\mathcal{S}, \mathcal{A}),\ \tilde{B}\tilde{Q}(s,a) - B\tilde{Q}(s,a) &\leqslant \epsilon_s^+ \\
\forall (s,a) \in (\mathcal{S}, \mathcal{A}),\ B\tilde{Q}(s,a) - \tilde{B}\tilde{Q}(s,a) &\leqslant \epsilon_s^-
\end{aligned}$$

or more compactly:

$$\forall (s,a) \in (\mathcal{S}, \mathcal{A}),\ -\epsilon_s^- \leqslant \tilde{B}\tilde{Q}(s,a) - B\tilde{Q}(s,a) \leqslant \epsilon_s^+$$

Definition 8. *Let $C_{B\tilde{Q}}$ a Hölder constant such that:*

$$C_{B\tilde{Q}} d(s,a,s',a') \leqslant |\tilde{Q}(s,a) - B\tilde{Q}(s,a)|, \forall (s,a) \in (\mathcal{S},\mathcal{A}),$$

Lemma 9. *Let $\epsilon_d + \epsilon_s^-$ bound the one step Bellman underestimation error in the solution of the NP-ALP. Then $\epsilon_d \leqslant d_{\max}^{\alpha}(C_{B\tilde{Q}} + C_{\tilde{Q}})$, where $d_{\max}$ is the maximum distance from a non-sampled state-action pair to the closest sampled state-action pair*[8]*.*

Proof. For a state-action where a Bellman constraint is present $\tilde{B}\tilde{Q}(s',a') = \tilde{Q}(s',a')$, and from definition 7 the maximum underestimation error is ϵ_s^-. Let there be some state-action (s,a) for which the Bellman constraint is missing, and let (s',a') be its nearest neighbor for which a Bellman constraint is present. Then:

$$\begin{aligned}
B\tilde{Q}(s,a) &\leqslant B\tilde{Q}(s',a') + C_{B\tilde{Q}} d(s,a,s',a') \\
&\leqslant \tilde{Q}(s',a') + \epsilon_s^- + C_{B\tilde{Q}} d^{\alpha}(s,a,s',a') \\
&\leqslant \tilde{Q}(s,a) + C_Q d^{\alpha}(s,a,s',a') + \epsilon_s^- + C_{B\tilde{Q}} d^{\alpha}(s,a,s',a') \\
&\leqslant \tilde{Q}(s,a) + \epsilon_s^- + d_{\max}^{\alpha}(C_{B\tilde{Q}} + C_{\tilde{Q}}) \\
\Rightarrow B\tilde{Q}(s,a) - \tilde{Q}(s,a) &\leqslant \epsilon_d + \epsilon_s^-
\end{aligned}$$

□

Lemma 10. *Let $\epsilon_C + \epsilon_s^+$ bound the one step Bellman overestimation error in the NP-ALP. Then for $C_{\tilde{Q}} > 0$:*

$$\epsilon_C \leqslant \max\left(0, (Q_{\max} - Q_{\min})\left(\frac{C_{B\tilde{Q}}}{C_{\tilde{Q}}} - 1\right)\right) \tag{3.6}$$

[8] If we assume that we can sample every action in each state instead of individual actions, the equation above becomes: $\epsilon_d \leqslant d_{\max}^{\alpha}(C_{B\tilde{V}} + C_{\tilde{V}})$ where in this case $d_{\max}$ is the maximum distance from a non-sampled state to the closest sampled state.

Proof. Let there be some state-action (s, a) that is constrained by a Hölder continuity constraint from another state-action (s', a'), such that its value is $\tilde{Q}(s, a) = \tilde{Q}(s', a') - C_{\tilde{Q}} d^{\alpha}(s, a, s', a')$. Then we have that $d^{\alpha}(s, a, s', a') \leqslant \frac{Q_{\max} - Q_{\min}}{C_{\tilde{Q}}}$ (otherwise we would have $Q(s, a) < Q_{\min}$), a Bellman constraint must be active for (s', a') such that $\tilde{B}\tilde{Q}(s', a') = \tilde{Q}(s', a')$ and from definition 7, $B\tilde{Q}(s', a) \geqslant \tilde{Q}(s', a) - \epsilon_s^+$. Consequently:

$$\begin{aligned} B\tilde{Q}(s, a) &\geqslant B\tilde{Q}(s', a') - C_{B\tilde{Q}} d^{\alpha}(s, a, s', a') \\ &\geqslant \tilde{Q}(s', a') - \epsilon_s^+ - C_{B\tilde{Q}} d^{\alpha}(s, a, s', a') \\ &= \tilde{Q}(s, a) + C_{\tilde{Q}} d^{\alpha}(s, a, s', a') - \epsilon_s^+ - C_{B\tilde{Q}} d^{\alpha}(s, a, s', a') \\ \Rightarrow \tilde{Q}(s, a) - B\tilde{Q}(s, a) &\leqslant \epsilon_s^+ + d^{\alpha}(s, a, s', a')(C_{B\tilde{Q}} - C_{\tilde{Q}}). \end{aligned}$$

For $C_{B\tilde{Q}} \geqslant C_{\tilde{Q}}$ the above is maximized for $d^{\alpha}(s, a, s', a') = \frac{Q_{\max} - Q_{\min}}{C_{\tilde{Q}}}$ yielding:

$$\epsilon_C = (Q_{\max} - Q_{\min}) \left(\frac{C_{B\tilde{Q}}}{C_{\tilde{Q}}} - 1 \right),$$

while otherwise it is maximized for $d^{\alpha}(s, a, s', a') = 0$ yielding:

$$\epsilon_C = 0.$$

□

We are now ready to state the main theorem of this section:

Theorem 11. *Let $\tilde{V}$ be the solution to the NP-ALP. The return $\tilde{V}^{\pi}$ from the greedy policy over $\tilde{V}$ satisfies:*

$$\forall s \in \mathcal{S},\ \tilde{V}^{\pi}(s) \geqslant V^*(s) - \frac{\epsilon_C + \epsilon_d + \epsilon_s^- + \epsilon_s^+}{1 - \gamma}$$

Proof. From lemmata 9 and 10 we have that

$$\forall (s, a) \in (\mathcal{S}, \mathcal{A}),\ \ -\epsilon_d - \epsilon_s^- \leqslant \tilde{Q}(s, a) - B\tilde{Q}(s, a) \leqslant \epsilon_C + \epsilon_s^+,$$

and the result follows directly from theorem 6.

□

Lemma 14 below allows us to bound the Hölder constant of $B\tilde{V}$, in terms of the Hölder constant of the reward and transition functions, while lemma 15 bounds how large $C_{\tilde{V}}$ needs to be in order to guarantee $\epsilon_C = 0$. Note that while a Hölder continuous reward and transition function implies a Hölder continuous $B\tilde{V}$, it is not a requirement. One could easily come up with discontinuous reward and transition functions that still result in continuous value functions.

Definition 12. *If the reward function is C_r-Hölder continuous, it satisfies the following constraint for every two states s_1 and s_2:*

$$|r(s_1, a) - r(s_2, a)| \leqslant C_r d(s_1, s_2)^\alpha$$

Definition 13. *If the transition model is C_p-Hölder continuous it satisfies the following constraint for every two states s_1 and s_2, and all V with $C_V = 1$:*

$$\Big| \int_{s'} (p(s'|s_1, a) - p(s'|s_2, a))V(s')ds' \Big| \leqslant C_p d(s_1, s_2)^\alpha$$

Observe that this bounds the difference in expected next state values with respect to a normalized V. If $C_V \neq 1$, the worst case difference can be scaled appropriately.

Lemma 14.

$$C_{B\tilde{Q}} \leqslant C_r + \gamma C_p C_{\tilde{V}} \tag{3.7}$$

Proof. Follows directly from the definitions of C_r, C_p, $C_{\tilde{V}}$ and $C_{\tilde{Q}}$. □

Lemma 15. *If $\gamma C_p < 1$ and $C_{\tilde{Q}} \geqslant \frac{C_r}{1-\gamma C_p}$, $\epsilon_C = 0$.*

Proof. The result follows directly by substituting equation 3.7 in equation 3.6 and requiring $\frac{C_r + \gamma C_p C_{\tilde{V}}}{C_{\tilde{Q}}} \leqslant 1$. □

Note that $\gamma C_p < 1$ is satisfied is many noise models, e.g., actions that add a constant impulse with Gaussian noise.

3.7 Using noisy samples

Ideally, we would want samples that provide the full probability distribution for a given state-action pair. Unfortunately these samples can be very difficult or impossible to obtain in most realistic domains. Instead, what we'll have are samples from the probability distribution. Given that in a continuous state space it may be statistically impossible to visit states more than once, we'll only have a single instantiation of the transition probabilities for every sampled state-action pair, providing a very unreliable approximation to Bellman's equation.

One way to deal with noisy transitions is by averaging over multiple repetitions. In a discrete environment with very few states and actions, we would sample each state-action enough times to be certain that our estimates are close to the true values with high probability. As we will see, there is a principled way to do the same for Hölder continuous value functions.

3.7.1 k nearest neighbors

Given that the value function is Hölder continuous, the value of any state-action pair can be expressed in terms of any other state-action pair as $Q(s_j, a_j) = Q(s_i, a_i) + \xi_{ij} C_{Q^*} d_{ij}^{\alpha}$, where $d_{ij}^{\alpha} = d(s_j, a_j, s_i, a_i)^{\alpha}$ and ξ_{ij} is a fixed but possibly unknown constant in $[-1, 1]$. For sample (s_i, a_i, r_i, s_i'), define:

$$x_{(s_i,a_i,r_i,s_i'),j} = r_i + \gamma V(s_i') + \xi_{ij} C_{Q^*} d_{ij}^{\alpha}.$$

Then:

$$\begin{aligned} E_{s_i'}[x_{(s_i,a_i,r_i,s_i'),j}] &= E_{s_i'}[r_i + \gamma V(s_i')] + \xi_{ij} C_{Q^*} d_{ij}^{\alpha} \\ &= Q(s_i, a_i) + \xi_{ij} C_{Q^*} d_{ij}^{\alpha}. \end{aligned}$$

Consider a sampled state-action pair (s_0, a_0) and its $k-1$ nearest neighbors (s_i, a_i) for $i = 1, \ldots, k-1$. Setting $\xi_{ij} = -1 \; \forall \; i, j$, we can arrive at a pessimistic estimate for its value by averaging over the predicted value for that state-action from the sample itself and all its neighbors: $\hat{Q}(s_0, a_0) = \frac{\sum_{i=0}^{k-1} x_{(s_i, a_i, r_i, s_i'), 0}}{k}$.

In the LP, this amounts to substituting every Bellman constraint with the distance adjusted average of the Bellman constraints for the k nearest neighbors. The NP-ALP becomes:

$$
\begin{aligned}
&\textit{minimize} \; \sum_{s \in \tilde{S}} \tilde{V}(s) \\
&\textit{subject to}: \\
&(\forall (s, a) \in \tilde{S}) \tilde{V}(s) \geqslant \sum_{i=0}^{k-1} \Big(R(s_i, a_i, s_i') + \gamma \tilde{V}(s_i') - C_{\tilde{Q}} d^{\alpha}(s_0, a_0, s_i, a_i) \Big) \\
&\tilde{V} \in \mathcal{M}_{C_{\tilde{V}}, \alpha}
\end{aligned}
$$

The next subsection shows that as the density and number of neighbors k increase, the estimate will converge to the true value $Q^*(s_0, a_0)$.

3.7.2 Finite sample behavior and convergence

We would like to bound the errors ϵ_s^- and ϵ_s^+ introduced to the LP by approximating Bellman's equation. We will decompose the error into two pieces: the error caused by using a finite number of neighbors, and the error caused by using neighbors at a non-zero distance from the point of interest.

From Hoeffding's inequality, we have that for every sampled Bellman constraint the probability of the mean over k samples being more than t away from the expectation is bounded by:

$$
P(|\hat{B}Q_i - BQ_i| \geqslant t) \quad \leqslant \quad 2e^{-\frac{2t^2 k}{(Q_{\max} - Q_{\min})^2}},
$$

Note that the values returned by the LP will always lie in $[Q_{\min}, Q_{\max}]$ (see section 3.1 for the definition of $Q_{\max}$ and $Q_{\min}$).

From the union bound, we have that the probability δ of the mean over k samples being more than t away in any of the n samples, is no more than the sum of the individual probabilities:

$$\delta \leqslant n2e^{-\frac{2t^2k}{(Q_{\max}-Q_{\min})^2}}$$

Taking logarithms on both sides and solving for t, we have that for a given probability of failure δ, the absolute error is upper bounded by:

$$t \leqslant \frac{(Q_{\max}-Q_{\min})}{\sqrt{2}}\sqrt{\frac{\ln\frac{2n}{\delta}}{k}}$$

Assuming that n Bellman constraints are spread uniformly (up to a constant C_s) across the state-action space, the volume contained by the minimum hypersphere containing k points is proportional to $\frac{k}{n}$. The radius of that hypersphere r_k is related to that volume as: $\frac{k}{n} = cr_k^D$, where D is the dimensionality of the space (ambient or underlying manifold). Thus the error introduced by using neighbors at a non-zero distance from the point of interest will be upper bounded by $c_s\left(\frac{k}{n}\right)^{\frac{\alpha}{D}}$.

From all the above, we have for ϵ_s^- and ϵ_s^+ with probability $1-\delta$:

$$\epsilon_s^- \leqslant \frac{(Q_{\max}-Q_{\min})}{\sqrt{2}}\sqrt{\frac{\ln\frac{2n}{\delta}}{k}} + C_sC_{\tilde{Q}}\left(\frac{k}{n}\right)^{\frac{\alpha}{D}}$$

$$\epsilon_s^+ \leqslant \frac{(Q_{\max}-Q_{\min})}{\sqrt{2}}\sqrt{\frac{\ln\frac{2n}{\delta}}{k}}$$

In addition, for the assumptions above we have for ϵ_d:

$$\epsilon_d \leqslant C_sC_{\tilde{Q}}\left(\frac{1}{n}\right)^{\frac{\alpha}{D}}.$$

Corollary 16. *Setting* $k = \left(\ln \frac{2n}{\delta}\right)^{\frac{D}{2\alpha+D}} n^{\frac{2\alpha}{2\alpha+D}}$ *we have that the bound from theorem 11 becomes:*

$$\begin{aligned} \forall s \in \mathcal{S},\ \tilde{V}^{\pi}(s) \ &\geqslant\ V^{*}(s) - \frac{\epsilon_c + C_s C_{\tilde{Q}} \left(\frac{1}{n}\right)^{\frac{\alpha}{D}}}{1-\gamma} \\ &- \frac{\left(2\frac{(Q_{\max}-Q_{\min})}{\sqrt{2}} + C_s C_{\tilde{Q}}\right)\left(\frac{\ln \frac{2n}{\delta}}{n}\right)^{\frac{\alpha}{2\alpha+D}}}{1-\gamma} \end{aligned}$$

w. p. $1-\delta$, *and* $\tilde{V}^{\pi}(s) \to V^{*}(s) - \frac{\epsilon_c}{1-\gamma}$ *as* $n \to \infty$.

4

Related Work

4.1 Approximate Linear Programming

In the realm of parametric value function approximation, Regularized Approximate Linear Programming (RALP) (Petrik et al., 2010) is probably the most closely related line of work to NP-ALP. As a parametric method, RALP down-selects from a potentially large initial set of features via $l1$-regularization on the feature weights.

Compared with NP-ALP, RALP's main strength is that for aggressive settings of its regularization parameter, the final set of feature weights may be smaller than the number of samples corresponding to basic variables in NP-ALP.

On the other hand, NP-ALP has many advantages over RALP. While RALP helps relieve the burden of feature selection, the user still has to decide which features to include in the feature pool. In contrast NP-ALP requires only the definition of a distance function which we find more straightforward for many domains.

In its initial version, RALP required samples of the full Bellman equation, or a domain with no or very little noise. Recent improvements (Taylor and Parr, 2012) have lifted this limitation, although now a smoothing kernel has to be specified in addition to the feature pool. NP-ALP deals with function approximation and noise

in a unified manner through the distance function.

Another advantage NP-ALP has over RALP and other ALP methods (de Farias and Van Roy, 2003) is that its bounds are much stronger. NP-ALP provides the first known max-norm bounds for any form of ALP, as well as the first finite sample complexity results for real world samples (not samples from the full Bellman equation).

Finally, an advantage of NP-ALP is that it incorporates large and infinite action spaces very naturally, while traditional ALP methods require significant extensions for large action spaces (Pazis and Parr, 2011).

4.2 Feature-free methods

A number of papers in Reinforcement Learning have explored the idea of dispensing completely with features. Since all but the smallest toy problems are too large to be solved via naive, uniform discretization, these methods have concentrated on finding effective ways to discretize the state space in a non-uniform manner.

An early example is the work of Munos and Moore (2002), where they evaluate different criteria of splitting the state space, albeit assuming that a model of the dynamics and reward function is available, and that the dynamics are deterministic.

A more recent line of work, which makes significantly more realistic assumptions and does not require features is Fitted Q-Iteration (FQI) with tree-based approximators (Ernst et al., 2005). To date, FQI with random forests has come closer to a true black box approach than most batch mode RL algorithms. When combined with extra trees in particular, a random forest based approximator that seems to work well in practice, its only parameters are the number of trees, the number of random cuts from which to choose each decision point, the number of samples per leaf, and the number of iterations to perform on the outer value iteration loop. One potential drawback is that similarly to most other policy and value iteration methods,

handling large action spaces can be prohibitively expensive both during execution and training without modifications to the learning process (Pazis and Lagoudakis, 2011). In addition, compared with NP-ALP, the random forest solution may require significantly more memory to store. Finally, the resulting value function is piecewise constant; this may not cause such gross overestimation errors of unvisited areas of the state-action space as parametric methods, but is still not very safe in risky domains where we may not have samples for obviously poor areas. This is in contrast with the pessimistic aversion of NP-ALP towards unexplored state-actions.

4.3 Kernelized methods

Parametric methods often have a non-parametric "dual" equivalent, inspired by analogous supervised learning techniques. These methods fall into the non-parametric category, because even though we have to define a kernel "shape", the location of each kernel is defined by the samples. So far, from the three standard solution methods used in approximate RL, policy and value iteration are the ones that have seen kernelized implementation.

Taylor and Parr (2009) provide a good overview of the field and offer a unifying approach to a number of independently developed kernelized algorithms. Farahmand et al. (2009) are notable for including sample complexity results and max-norm error bounds, but their bounds depend upon difficult to measure quantities, such as concentrability coefficients.

In general, kernelized approaches associated with policy iteration or value iteration tend to require more restrictive and complicated assumptions yet provide weaker guarantees. In addition, picking an appropriate kernel is often difficult and less intuitive than picking a good distance function. Finally, such methods tend to be at a disadvantage when dealing with large action spaces and seem hard to get to work well in practice.

4.4 Kernel based methods

Similarly with the kernelized methods, in kernel based methods one has to specify a kernel shape but the location of each kernel is defined by the samples. The general principle of these methods is that the value of each state-action is a weighted average over the values of (often all) other samples, with the contribution of each sample depending on the kernel function.

One such algorithm is "Kernel-Based Reinforcement Learning" (Ormoneit and Sen, 2002) which is a consistent RL algorithm for continuous state spaces, based on kernel averaging. A more recent example is the work of Kroemer and Peters (2012) who also use kernel density estimates and Kveton and Theocharous (2012) who use cover trees to select a representative set of states.

In general, kernel based methods suffer from some of the same problems that kernelized approaches do, although they do seem easier to get working in practice.

5

Experimental Results

We tested NP-ALP in noisy versions of three popular domains, "Inverted pendulum regulator" (continuous action space), "Car on the hill" (discrete action space) and "bicycle balancing" (continuous 2D action space).

We should note that since both the model and a vast amount of accumulated knowledge exist for these domains, many algorithms exist that achieve good performance when taking advantage of this information. The purpose of these experiments is not to claim that policies produced by NP-ALP outperform policies produced by such algorithms. Instead our goal is twofold: To demonstrate that we can tackle these problems even under the weakest of assumptions, with algorithms that provide strong theoretical guarantees, providing some indication that NP-ALP would be able to perform well on domains where no such knowledge exists, and to show that the performance achieved by our algorithms supports that which is predicted by our bounds.

5.1 Inverted Pendulum

The inverted pendulum problem (Wang et al., 1996) requires balancing a pendulum of unknown length and mass at the upright position by applying forces to the cart to which it is attached. The 2-dimensional continuous state space includes the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. We standardized the state space and used PCA, keeping only the first principal component. The *continuous* action space of the process is the range of forces in $[-50N, 50N]$. We repeated our experiments for two different levels of noise added to each action selected by the agent. One set of experiments was performed with uniform noise in $[-10N, 10N]$, and one in $[-20N, 20N]$ (significantly more than typical for this domain).

Many researchers in reinforcement learning choose to approach this domain as an avoidance task, with zero reward as long as the pendulum is above the horizontal configuration and a negative reward when the controller fails. Instead we approached the problem as a regulation task, where we are not only interested in keeping the pendulum upright, but we want to do so while minimizing the amount of force we are using. Thus a reward of $1 - (a/50)^2$, was given as long as $|\theta| \leqslant \pi/2$, and a reward of 0 as soon as $|\theta| > \pi/2$, a property which also signals the termination of the episode. The discount factor of the process was set to 0.98 and the control interval to 100ms. Coupled with the high levels of noise, making full use of the available continuous action range is required to get good performance in this setting.

Unless noted otherwise, for all experiments below, $C_{\tilde{Q}} = C_{\tilde{V}} = 1.5$, $\alpha = 1$ and the distance function was set to the two norm of the difference between state-actions, with the action space rescaled to $[-1, 1]$. Training samples were collected in advance by starting the pendulum in a randomly perturbed state close to the equilibrium state $(0, 0)$ and following a policy which selected actions uniformly at random.

Figure 5.1 shows a histogram overlay of the actions chosen by a representa-

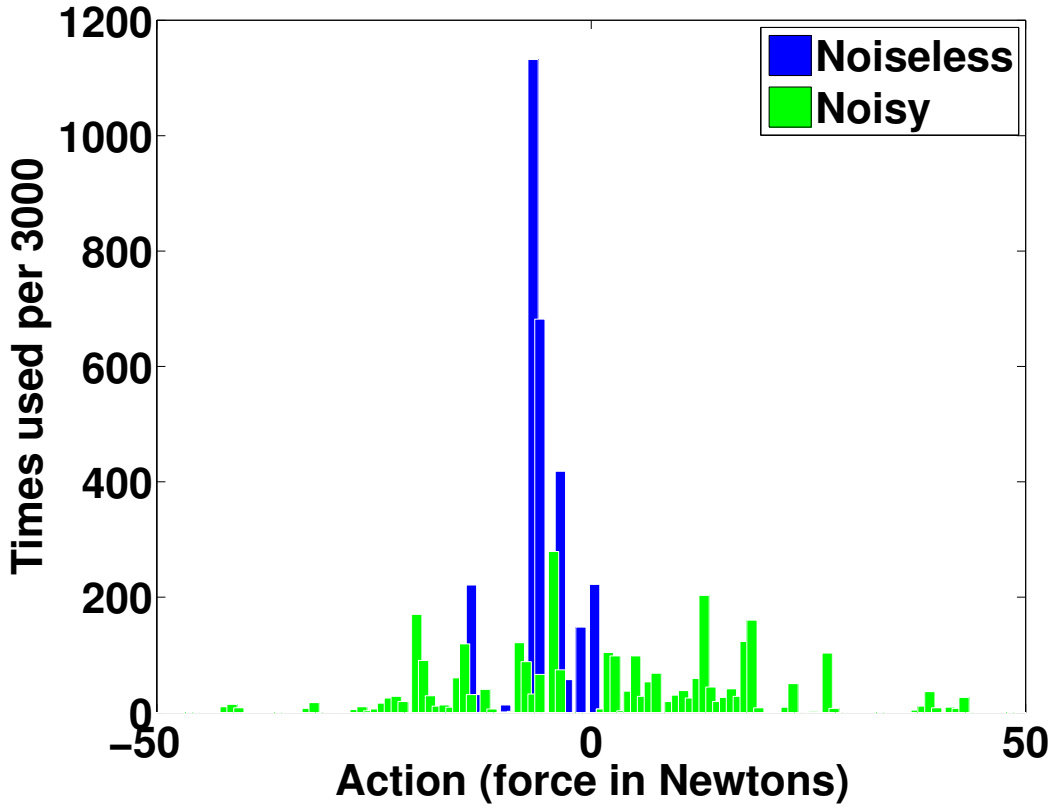
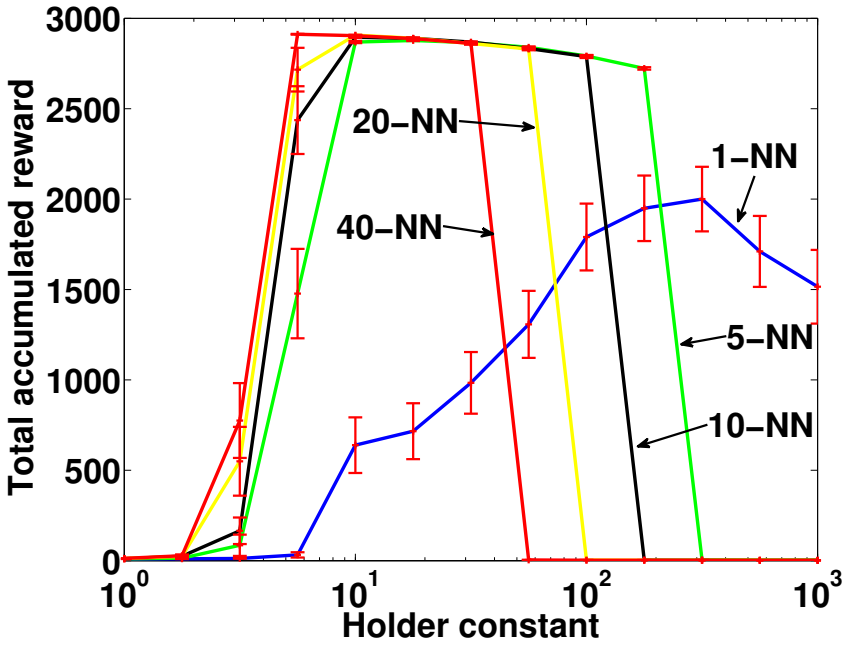
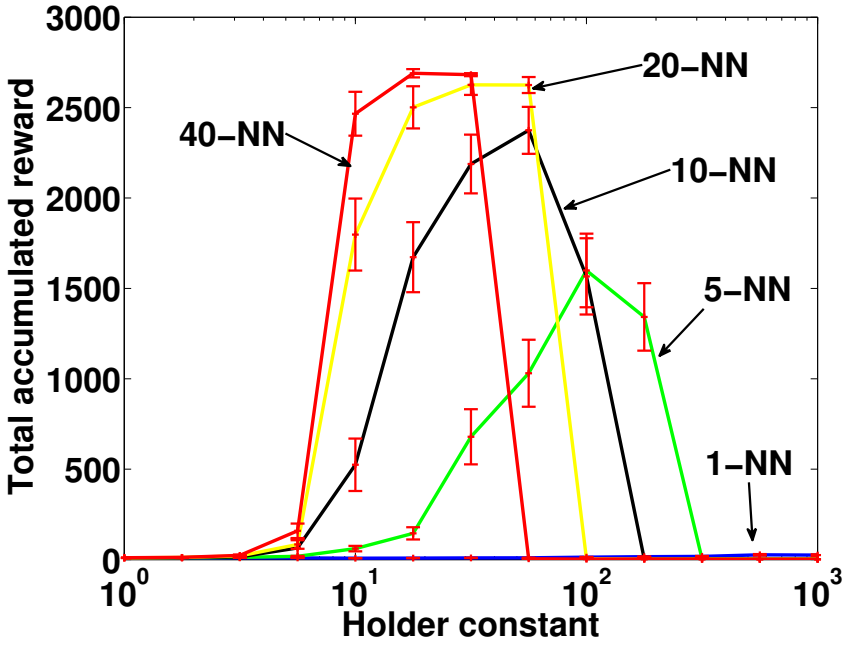


FIGURE 5.1: Inverted pendulum regulator: Histogram overlay of actions chosen be a representative policy trained and executed *without (blue) and* with (green) uniform noise in $[-20, 20]$.

tive policy trained and executed *without (blue) and* with (green) uniform noise in $[-20N, 20N]$. One can see that when there is no noise the controller is able to balance the pendulum with a minimum amount of force, while when there is a significant amount of noise, a much wider range of actions is used to keep the pendulum from falling. In both cases, the controller is able to make very good use of the continuous action range. Also notice that the histograms have "gaps". Even though the controllers have access to the full continuous actions range, only actions that have been sampled during the training phase at a state close to the current state can ever be selected. Due to the pessimistic nature of the action selection mechanism, all other actions are assumed to have worse values.



(a)



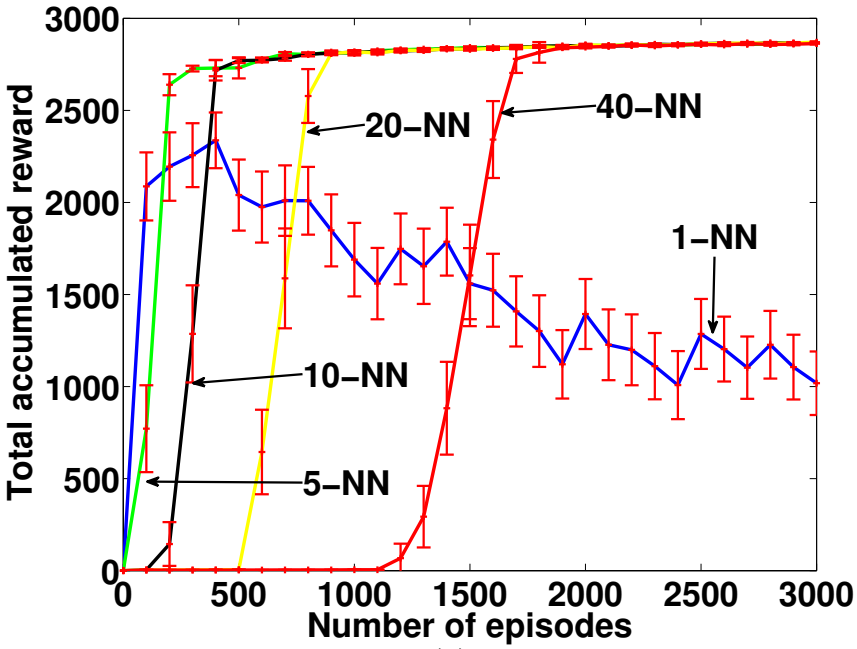
(b)

FIGURE 5.2: Inverted pendulum regulator (a), (b): Total accumulated reward per episode versus the Hölder constant, with uniform noise in $[-10, 10]$ (a) and $[-20, 20]$ (b). Averages and 95% confidence intervals are over 100 independent runs.

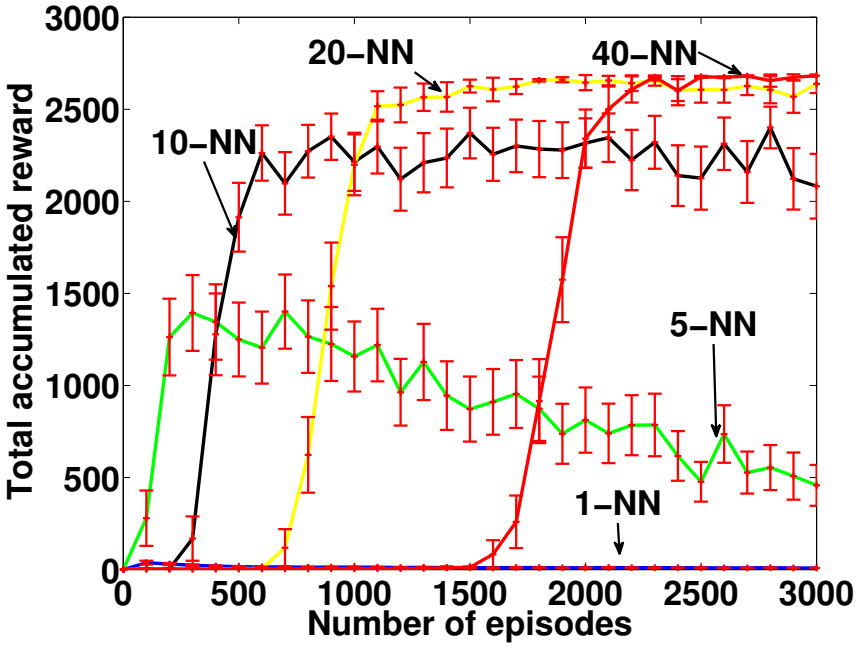
Figure 5.2 shows total accumulated reward versus the Hölder constant with uniform noise in $[-10, 10]$ (a) and $[-20, 20]$ (b), for 3000 training episodes, averaged over 100 independent training runs, along with 95% confidence intervals. Notice the logarithmic scale on the x axis. We can see that the shape of the graphs reflects that of the bounds. When $C_{\tilde{Q}}$ is too small, ϵ_C is large, while when $C_{\tilde{Q}}$ is too large, ϵ_d is large. Additionally, for small values of k, ϵ_s^- and ϵ_s^+ are large. One interesting behavior not directly explained by the bounds is that for small values of k, the best performance is achieved for large values of $C_{\tilde{Q}}$. We believe that this is because the larger $C_{\tilde{Q}}$ is, the smaller the area affected by each overestimation error. Comparing the two graphs, we can see that different values of k exhibit much greater performance overlap over $C_{\tilde{Q}}$ for smaller amounts of noise.

Figure 5.3 shows the total accumulated reward as a function of the number of training episodes with uniform noise in $[-10, 10]$ (a) and $[-20, 20]$ (b). Again the observed behavior is the one expected from our bounds. While larger values of k ultimately reach the best performance even for high levels of noise, the $C_s C_{\tilde{Q}} \left(\frac{k}{n}\right)^{\frac{\alpha}{D}}$ component of ϵ_s^- and $C_s C_{\tilde{Q}} \left(\frac{1}{n}\right)^{\frac{\alpha}{D}}$ of ϵ_d penalize large values of k when n is not large enough. In addition (perhaps unintuitively), for any constant k, increasing the number of samples beyond a certain point increases the probability that ϵ_s^+ will be large for some state ($\max_s \epsilon_s^+$), causing a decline in average performance and increasing variance. Thus, in practical applications the choice of k has to take into account the sample density and the level of noise. Comparing the two graphs, we can see that this phenomenon is more pronounced at higher noise levels, affecting larger values of k.

Before we move on to the next domain, we should highlight the differences and advantages of NP-ALP over other methods which have been applied to numerous variations of the inverted pendulum problem. The first difference is that NP-ALP



(a)



(b)

FIGURE 5.3: Inverted pendulum regulator (a), (b): Total accumulated reward per episode versus number of training episodes, with uniform noise in $[-10, 10]$ (a) and $[-20, 20]$ (b). Averages and 95% confidence intervals are over 100 independent runs.

allows the use of a continuous action space, a necessary property for many realistic domains/reward functions, which NP-ALP shares with very few other reinforcement learning algorithms. In addition, the few RL algorithms supporting continuous actions offer weaker if any performance guarantees. Finally, approaches from control theory and operations research require much stronger assumptions and knowledge about the domain, which may not be available in other interesting domains.

5.2 Car on the hill

The car on the hill problem (Ernst et al., 2005) involves driving an underpowered car stopped at the bottom of a valley between two hills, to the top of the steep hill on the right. The 2-dimensional continuous state space (p, v) includes the current position p and the current velocity v. The controller's task is to (indirectly) control the acceleration using a thrust action $u \in \{-4, 4\}$, under the constraints $p > -1$ and $|v| \leqslant 3$ in order to reach $p = 1$. The task requires temporarily driving away from the goal in order to gain momentum. The agent receives a reward of -1, if a constraint is violated, a reward of $+1$, if the goal is reached, and a zero reward otherwise. The discount factor of the process was set to 0.98 and the control interval to 100ms.

While this domain is usually modeled as noise free, we chose to add uniform noise in $[-2, 2]$ to make the problem more challenging. The distance function was chosen to be the two norm of the difference between state-actions, while scaling the speed portion of the state from $[-3, 3]$ to $[-1, 1]$, and the action space from $[-4, 4]$ to $[-1, 1]$.

Training samples were collected in advance from "random episodes", that is, starting the car in a randomly perturbed state and following a policy which selected actions uniformly at random. Each episode was allowed to run for a maximum of 200 steps or until a terminal state was reached, both during sampling and policy execution.

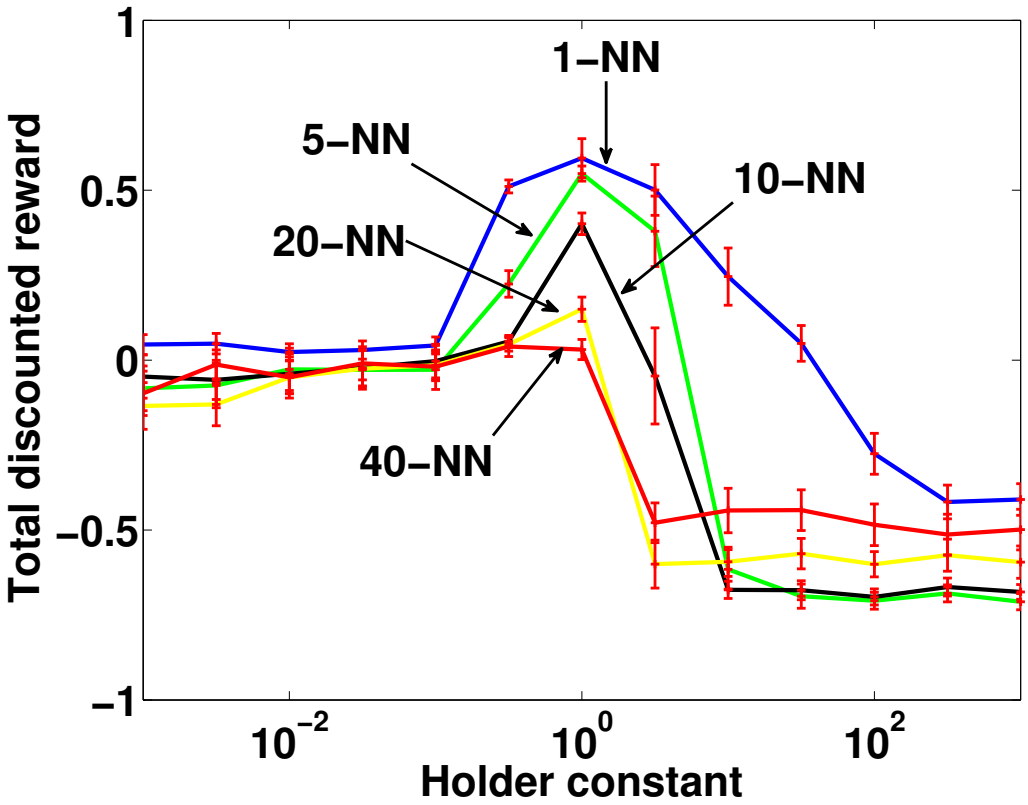


FIGURE 5.4: Car on the hill: Total accumulated reward versus the Hölder constant. Averages and 95% confidence intervals are over 100 independent runs.

Figure 5.4 shows total accumulated discounted reward versus the Hölder constant for 200 training episodes. As we can see, this domain is not very much affected by noise. For 200 training episodes, the one nearest neighbor controller performs best, while larger values of k require more samples before they achieve good performance.

5.3 Bicycle Balancing

The bicycle balancing problem (Ernst et al., 2005), has four state variables (angle θ and angular velocity $\dot{\theta}$ of the handlebar as well as angle ω and angular velocity $\dot{\omega}$ of the bicycle relative to the ground). The *continuous* action space is two dimensional and consists of the torque applied to the handlebar $\tau \in [-2, +2]$ and the displacement

of the rider $d \in [-0.02, +0.02]$. Noise comes in the form of a uniform component in $[-0.02, +0.02]$ added to the displacement portion of the action space. The goal is to prevent the bicycle from falling.

Again we approached the problem as a regulation task, rewarding the controller for keeping the bicycle as close to the upright position as possible. A reward of $1 - |\omega| \times (\pi/15)$ was given as long as $|\omega| \leqslant \pi/15$, and a reward of 0 as soon as $|\omega| > \pi/15$, which also signals the termination of the episode. The discount factor was 0.98, the control interval was 10ms and training trajectories were truncated after 20 steps. We standardized the state space and used PCA, keeping only the first principal component. The distance function was the two norm of the difference between state actions, with each dimension of the action space rescaled to $[-1, 1]$.

Figure 5.5 shows total accumulated reward versus the Hölder constant for 100 training episodes. To our surprise this domain turned out to be the easiest of the three. Except for the one nearest neighbor controller, all others perform well for a large range of constants. Even though for large values of the Hölder constant the controllers end up being very pessimistic, they are able to select actions that balance the bicycle almost all of the time.

The one nearest neighbor controller has the expected behavior with performance degrading for larger Hölder constants, while all other controllers achieve excellent performance for all but the smallest Hölder constants.

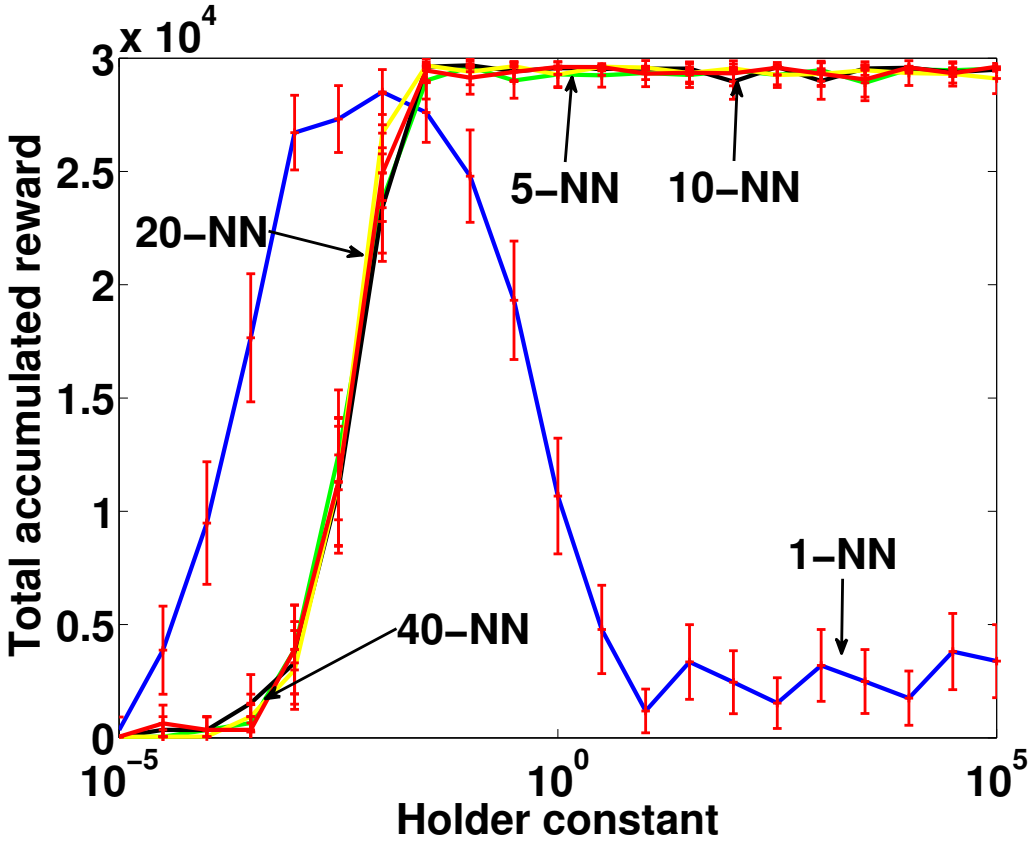


FIGURE 5.5: Bicycle balancing: Total accumulated reward versus the Hölder constant. Averages and 95% confidence intervals are over 100 independent runs.

6

Discussion and Conclusion

6.1 Strengths and weaknesses

NP-ALP and offers a number of important advantages over its feature based and tabular-representation based counterparts:

- The most obvious advantage is that because NP-ALP is non-parametric, there is no need to define features or perform costly feature selection.
- NP-ALP is amenable to problems with large (multidimensional) or even infinite (continuous) state and action spaces.
- NP-ALP offers significantly stronger and easier to compute performance guarantees than feature based ALP, the first (to the best of our knowledge) max-norm performance guarantees for any ALP algorithm.
- NP-ALP offers finite sample performance guarantees under mild assumptions, even in the case where noisy, real world samples are used instead of the full Bellman equation.

- Defining a distance function can be much easier and more intuitive than coming up with good features for parametric methods.

Of course, as with any method, NP-ALP is not without weaknesses. The first weakness, is that NP-ALP can be computationally demanding when the number of samples is large and the distance function of choice does not allow for pruning the majority of constraints. Nevertheless, iteratively adding samples to the linear program as explained in section 3.4 seems to work extremely well in practice.

The second weakness is the potential difficulty in selecting a good distance function. While we believe that selecting a distance function provides more flexibility and is easier as well as more intuitive than selecting a good set of features, taking advantage of this flexibility is by no means a trivial task.

6.2 Future work

In our experiments, samples were drawn from policies which selected actions uniformly at random. One thing that has become apparent is that such an approach to sampling cannot scale to larger domains. Sampling certain parts of the state space can be extremely difficult for a random policy, not to mention that if samples come from the real world, they can be very costly. In our experiments, the sampling policy ended up being the bottleneck, constraining the kinds of domains we could tackle. We are currently working on an optimistic approach, similar to the pessimistic approach used in this thesis, for principled exploration of high dimensional continuous spaces.

As noted above, NP-ALP's flexibility in selecting the distance function is simultaneously the source of it's greatest strength and weakness. Section 3.5 introduced an approach that can take advantage of the structure of the manifold, and reduce the burden of selecting a good global distance function from the designer. Although

a first step in the right direction, we believe there is still work to be done in automatic distance function selection before NP-ALP (or any other RL algorithm) can be considered a true black box approach.

In this thesis, we focused on the fully observable case. However in real life full observability is a luxury we seldomly have. In most domains our state information is both incomplete and noisy. Robust RL methods, that take partial observability and measurement noise into account and try to correct for them are an important next step.

NP-ALP does not depend on a model. While this is highly advantageous in many interesting domains where we don't have a model, we are also interested in non-naive methods of integrating prior knowledge of a domain. Such knowledge may come as partial and/or inaccurate knowledge of the reward and transition functions, as well as prior knowledge of bounds or the shape of the value function. Making use of such knowledge may help to significantly lower the number of samples required in order to achieve good performance.

Bibliography

de Farias, D. P. and Van Roy, B. (2003), "The Linear Programming Approach to Approximate Dynamic Programming," *Operations Research*, 51, 850–865.

de Farias, D. P. and Van Roy, B. (2004), "On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming," *Mathematics of OR*, 29, 462–478.

Ernst, D., Geurts, P., and Wehenkel, L. (2005), "Tree-Based Batch Mode Reinforcement Learning," *Journal of Machine Learning Research*, 6, 503–556.

Farahmand, A., Ghavamzadeh, M., Szepesvari, C., and Mannor, S. (2009), "Regularized policy iteration," *Advances in Neural Information Processing Systems*, 21, 441–448.

Hoeffding, W. (1963), "Probability Inequalities for Sums of Bounded Random Variables," *Journal of the American Statistical Association*, 58, pp. 13–30.

Kaelbling, L. P., Littman, M., and Moore, A. (1996), "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, 4, 237–285.

Kroemer, O. and Peters, J. (2012), "A non-parametric approach to dynamic programming," *Advances in Neural Information Processing Systems*, 24, to appear.

Kveton, B. and Theocharous, G. (2012), "Kernel-Based Reinforcement Learning on Representative States," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, (AAAI)*.

Lagoudakis, M. G. and Parr, R. (2003), "Least-Squares Policy Iteration," *Journal of Machine Learning Research*, 4, 1107–1149.

Munos, R. and Moore, A. (2002), "Variable resolution discretization in optimal control," *Machine Learning*, 49, 291–323.

Ormoneit, D. and Sen, Ś. (2002), "Kernel-based reinforcement learning," *Machine Learning*, 49, 161–178.

Pazis, J. and Lagoudakis, M. G. (2011), "Reinforcement Learning in Multidimensional Continuous Action Spaces," in *IEEE Symposium Series in Computational Intelligence 2011 (SSCI 2011)*, Paris, France.

Pazis, J. and Parr, R. (2011), "Generalized Value Functions for Large Action Sets," in *ICML-11*, pp. 1185–1192, ACM.

Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010), "Feature Selection Using Regularization in Approximate Linear Programs for Markov Decision Processes," in *ICML-10*, pp. 871–878, Haifa, Israel, Omnipress.

Puterman, M. L. (1994), *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley-Interscience.

Sutton, R. and Barto, A. (1998), *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Massachusetts.

Taylor, G. and Parr, R. (2009), "Kernelized value function approximation for reinforcement learning," in *ICML '09*, pp. 1017–1024, New York, NY, USA, ACM.

Taylor, G. and Parr, R. (2012), "Value Function Approximation in Noisy Environments Using Locally Smoothed Regularized Approximate Linear Programs," in *Proceedings of the Twenty-Eighth International Conference on Uncertainty in Artificial Intelligence (UAI-2012)*, Catalina Island, California, USA.

Wang, H., Tanaka, K., and Griffin, M. (1996), "An Approach to Fuzzy Control of Nonlinear Systems: Stability and Design Issues," *IEEE Transactions on Fuzzy Systems*, 4, 14–23.

Watkins, C. J. C. H. (1989), "Learning from Delayed Rewards," Ph.D. thesis, Cambridge University, Cambridge, United Kingdom.

Williams, R. and Baird, L. (1993), "Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions," Tech. rep., Northeastern University, College of Computer Science.