Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes

Sridhar Mahadevan
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
mahadeva@cs.umass.edu

Mauro Maggioni
Program in Applied Mathematics
Department of Mathematics
Yale University
New Haven, CT 06511
mauro.maggioni@yale.edu

July 17, 2006

Technical Report 2006-35
Department of Computer Science
140 Governors Drive
University of Massachusetts
Amherst, Massachusetts 01003-9624

Abstract

This paper introduces a novel paradigm for solving Markov decision processes (MDPs), based on jointly learning representations and optimal policies. Proto-value functions are geometrically customized task-independent basis functions forming the building blocks of all value functions on a given state space graph or manifold. In this first of two papers, proto-value functions are constructed using the eigenfunctions of the (graph or manifold) Laplacian, which can be viewed as undertaking a Fourier analysis on the state space graph. The companion paper (Maggioni and Mahadevan, 2006) investigates building proto-value functions using a multiresolution manifold analysis framework called diffusion wavelets, which is an extension of classical wavelet representations to graphs and manifolds. Proto-value functions combine insights from spectral graph theory, harmonic analysis, and Riemannian manifolds. A novel variant of approximate policy iteration, called representation policy iteration, is described, which combines learning representations and approximately optimal policies. Two strategies for scaling proto-value functions to continuous or large discrete MDPs are described. For continuous domains, the Nyström extension is used to interpolate Laplacian eigenfunctions to novel states. To handle large structured domains, a hierarchical framework is presented that compactly represents proto-value functions as tensor products of simpler proto-value functions on component subgraphs. A variety of experiments are reported, including perturbation analysis to evaluate parameter sensitivity, and detailed comparisons of proto-value functions with traditional parametric function approximators.

Keywords: Markov Decision Processes, Reinforcement learning, Spectral Graph Theory, Harmonic Analysis, Riemannian Manifolds.

Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes

Sridhar Mahadevan mahadeva@cs.umass.edu
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA

Mauro Maggioni mauro.maggioni@yale.edu
Program in Applied Mathematics
Department of Mathematics
Yale University
New Haven,CT,06510

Editor:

Abstract

This paper introduces a novel paradigm for solving Markov decision processes (MDPs), based on jointly learning representations and optimal policies. Proto-value functions are geometrically customized task-independent basis functions forming the building blocks of all value functions on a given state space graph or manifold. In this first of two papers, proto-value functions are constructed using the eigenfunctions of the (graph or manifold) *Laplacian*, which can be viewed as undertaking a Fourier analysis on the state space graph. The companion paper (Maggioni and Mahadevan, 2006) investigates building proto-value functions using a multiresolution manifold analysis framework called *diffusion wavelets*, which is an extension of classical wavelet representations to graphs and manifolds. Proto-value functions combine insights from spectral graph theory, harmonic analysis, and Riemannian manifolds. A novel variant of approximate policy iteration, called *representation policy iteration*, is described, which combines learning representations and approximately optimal policies. Two strategies for scaling proto-value functions to continuous or large discrete MDPs are described. For continuous domains, the Nyström extension is used to interpolate Laplacian eigenfunctions to novel states. To handle large structured domains, a hierarchical framework is presented that compactly represents proto-value functions as tensor products of simpler proto-value functions on component subgraphs. A variety of experiments are reported, including perturbation analysis to evaluate parameter sensitivity, and detailed comparisons of proto-value functions with traditional parametric function approximators.

Keywords: Markov decision processes, reinforcement learning, value function approximation, manifold learning, spectral graph theory.

1. Introduction

This paper introduces a novel framework for solving Markov decision processes (MDPs) (Puterman, 1994), by simultaneously learning both the underlying representation or basis functions and (approximate) optimal policies. The framework is based on a new type of basis representation for approximating value functions called a *proto-value function* (Mahadevan, 2005a,b). These are task-independent global basis functions that collectively span the space of all possible (square-integrable) value functions on a given state space. Proto-value functions incorporate geometric constraints intrinsic to the environment: states close in Euclidean distance may be far apart in data space when measured on the manifold (e.g. two states on opposite sides of a wall in a spatial navigation task). While there have been several attempts to fix the shortcomings of traditional function approximators to address the inherent nonlinear nature of value functions, such as the idea of building successor representations (Dayan, 1993) or applying methods from computer vision to detect non-linearities (Drummond, 2002), these approaches have lacked a broad theoretical framework and consequently been explored in the relatively narrow context of discrete MDPs. We show that by rigorously formulating the problem of value function approximation as *approximating real-valued functions on a graph or manifold*, a more general solution emerges that not only has broader applicability than these previous methods, but also enables a new class of algorithms for solving MDPs by jointly learning representations and policies.

In this first of two papers, proto-value functions are viewed formally in the "Fourier" tradition, by diagonalizing and using the eigenfunctions of a symmetric operator called the Laplacian (associated with a heat diffusion equation) on a graph or Riemannian manifold (Rosenberg, 1997). In the second paper (Maggioni and Mahadevan, 2006), this manifold framework is extended to a multi-resolution approach, where basis functions are constructed using the newly developed framework of *diffusion wavelets* (Coifman and Maggioni, 2004). Mathematically, the proposed framework takes coordinate-free objects such as graphs and constructs coordinate-based representations from a harmonic analysis of the state space *geometry*. Value functions are viewed as elements of a Hilbert space of functions on a graph or Riemannian manifold. Under rather general conditions, the eigenfunctions of the *Laplacian*, a self-adjoint (symmetric) operator on differentiable functions on the manifold (Rosenberg, 1997), form an orthonormal basis for this Hilbert space. In the discrete setting of graphs, spectral analysis of the graph Laplacian operator provides an orthonormal set of basis functions on the graph for approximating any (square-integrable) function on the graph (Chung, 1997).

This research builds on recent work on manifold and spectral learning (Tenenbaum et al., 2000; Roweis and Saul, 2000; Belkin and Niyogi, 2004). In particular, Belkin and Niyogi (2004) pioneered the study of the Laplacian in the context of semi-supervised learning. A major difference that distinguishes our work from other work in manifold learning is that our focus is solving Markov decision processes. While value function approximation in MDPs is related to regression on graphs (Niyogi et al., 2003) in that both concern approximation of real-valued functions on the vertices of a graph, value function approximation is fundamentally different since target values are not specified by a teacher, and are initially unknown and must be iteratively determined by finding the (approximate) fixed point of the Bellman backup operator.

The proposed framework exploits the property that while the value function for an MDP may appear discontinuous when represented in Euclidean space (because of nonlinearities like walls), in many cases of interest it is a smooth function on the graph or manifold associated with the state (or state-action) space. The eigenfunctions of the (graph) Laplacian form a natural basis for approximating such smooth functions. This can be quantified in terms of approximation properties in various function spaces. From the geometric point of view, it is well-known from the study of the Laplacian on a manifold that its eigenfunctions capture intrinsic properties like "bottlenecks" (Cheeger, 1970). In the discrete setting, these ideas find a natural counterpart: the *Cheeger constant* (Chung, 1997) quantifies the decomposability of graphs and can be shown to be intimately connected to the spectrum of the graph Laplacian. These ideas formalize recent work in hierarchical reinforcement learning on decomposing action spaces by finding bottlenecks in state spaces. The eigenfunctions of the Laplacian also provide a way to construct *geodesically* smooth global approximations of a value function. In other words, smoothness respects the edges of the graph: this property is crucial in approximating value functions that appear discontinuous when represented in the underlying (Euclidean) ambient space, but are nonetheless smooth on the manifold. The spectrum of the Laplacian also provides information on random walks on a graph, which again are closely connected to state space traversals by executing policies in MDPs and reinforcement learning.

Informally, proto-value functions can be viewed as a new way to formulate the problem of reinforcement learning in terms of "proto-reinforcement learning". In proto-RL, an agent learns representations that reflect its experience and an environment's *large-scale* geometry. Early stages of policy learning often result in exploratory random walk behavior which generates a large sample of transitions. Proto-reinforcement learning agents convert these samples into learned representations that reflect the agent's experience and an environment's large-scale geometry. An agent in a one-dimensional environment (e.g. the "chain" MDP in (Lagoudakis and Parr, 2003; Koller and Parr, 2000)) should "see" the world differently from an agent in a hybercube or torus environment (e.g. the "blockers" domain studied by (Sallans and Hinton, 2004)). The unexpected result from applying the coordinate-free approach is that Laplacian eigenfunctions and diffusion wavelets appear remarkably adept at value function approximation. Proto-value functions can be constructed using either an off-policy method or an on-policy method. In the off-policy approach, representations emerge from a harmonic analysis of a random walk diffusion process on the state space, regardless of the exploration policy followed in learning the state space topology. In the "on-policy" approach, they are formed by diagonalizing the transition matrix, or more precisely the *Green' s function* of a policy directly (Maggioni and Mahadevan, 2005). This first paper focuses on the off-policy approach.

One hallmark of Fourier analysis is that the basis functions are localized in frequency, but not in time (or space). In n-dimensional Euclidean space, the trigonometric functions $\{e^{i\langle\lambda,x\rangle}\}_{\lambda\in\mathbb{R}^n}$ are eigenvectors that diagonalize any linear time-invariant system (Mallat, 1989). These are localized to a particular frequency λ, but have support over the entire space. Similarly, the eigenfunctions of the graph Laplacian are localized in frequency by being associated with a specific eigenvalue λ, but their support is in general the whole graph. This global characteristic raises a natural computational concern: how can Laplacian bases be represented compactly in large discrete and continuous spaces? We will address this problem in several ways: large factored spaces, such as grids, hypercubes, and tori, lead naturally to *product spaces* for which the Laplacian bases can be constructed efficiently using *tensor products*. For continuous domains, by combining low-rank approximations and the *Nystrom* interpolation method, Laplacian bases can be constructed quite efficiently (Drineas and Mahoney, 2005). Finally, it is well-known from the work on hierarchical reinforcement learning (Barto and Mahadevan, 2003) that value functions are inherently decomposable by exploiting the hierarchical structure of many real-world tasks. Consequently, basis functions need only be defined for subgraphs over which subtasks are defined.

Wavelet analysis has emerged over the past decade as a powerful alternative framework to Fourier analysis (Daubechies, 1992; Mallat, 1989). Wavelets are basis functions with compact support and have been extensively explored for Euclidean domains, including the Haar bases and the Daubechies bases. *Diffusion wavelets* proposed by Coifman and Maggioni (2004) extend the framework of wavelet analysis to more general spaces, such as graphs and manifolds. The second companion paper (Maggioni and Mahadevan, 2006) extends the Laplacian approach using diffusion wavelet bases, which allow a compact multi-level representation of diffusion processes on manifolds and graphs (Coifman and Maggioni, 2004; Bremer et al., 2004). Diffusion wavelets provide an interesting alternative to global Fourier eigenfunctions for value function approximation, since they encapsulate all the traditional advantages of wavelets: basis functions have compact support, and the representation is inherently hierarchical since it is based on multi-resolution modeling of processes at different spatial and temporal scales. In terms of approximation theory, the class of functions efficiently represented by diffusion wavelets is much larger than that corresponding to eigenfunctions of the Laplacian, for example it includes functions which are generally smooth but less smooth on "thin regions", as compared to the class of functions which are globally, uniformly smooth.

The rest of the paper is organized as follows. Section 2 contains a brief description of previous work. A detailed description of the Markov decision process (MDP) model is given in Section 3, including methods for approximating the value function. Section 4 provides an overview of proto-value functions, using examples to convey the key ideas. Section 5 introduces the mathematics underlying proto-value functions, in particular the Laplacian on Riemannian manifolds and its discrete counterpart, spectral graph theory. Section 6 describes the representation policy iteration (RPI) algorithm, a novel variant of policy iteration that combines the learning of basis functions (representations) and policies to solve MDPs. Section 7 evaluates RPI on some simple discrete MDPs to provide some insight. Section 8 and Section 9 describes two ideas for scaling proto-value functions to large discrete factored and continuous domains, including tensor methods that exploit the properties of graph Laplacians on *product spaces*, and the *Nyström* extension for interpolating eigefunctions from sampled states to novel states. These sections also contain a detailed experimental analysis of RPI on large MDPs, including the *blockers* task (Sallans and Hinton, 2004), the inverted pendulum and the mountain car (Sutton and Barto, 1998) continuous control tasks. Finally, Section 10 discusses several extensions of the proposed framework to new areas.

2. History and Related Work

2.1 Value Function Approximation

In the 1950s, Samuel (1959) pioneered the use of parametric function approximation in machine learning: he implemented a program to play checkers that adjusted the coefficients of a fixed polynomial approximator so that values of states earlier in a game reflected outcomes experienced later during actual play (a heuristic form of what is now formally called temporal difference learning (Sutton, 1984, 1988)). Board states s were translated into k-dimensional feature vectors $\phi(s)$, where the features or basis functions $\phi : S \rightarrow \mathbb{R}^k$ were hand engineered (for example, a feature could be the "number of pieces"). Although Samuel's ideas were foundational, research over the subsequent five decades has substantially revised and formalized his early ideas, principally by combining the mathematical model of Markov decision processes (MDPs) (Puterman, 1994) with theoretical and algorithmic insights from statistics and machine learning (Hastie et al., 2001). This body of research has culminated in the modern fields of approximate dynamic programming (ADP) and reinforcement learning (RL) (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998).

Classical methods for solving Markov decision processes (MDPs) have been studied for almost 50 years in operations research (OR) (Puterman, 1994). The principal methods that have been investigated are *value iteration*, *policy iteration*, and *linear programming*, where the value function is usually represented exactly using an orthonormal unit vector basis. More recently, these methods have also been extended to use approximations, such as least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003), approximate dynamic programming using linear programming (Farias, 2003; Guestrin et al., 2003), and least-squares temporal-difference learning (Bradtke and Barto, 1996; Boyan, 1999; Nedic and Bertsekas, 2003). These approximate methods can be viewed as projecting the exact value function onto a subspace spanned by a set of basis functions. The majority of this research has assumed the basis functions are hand engineered: we propose to construct the basis functions automatically.

Value function approximation has been studied by many researchers. Bertsekas and Tsitsiklis (1996) provide an authoritative review. Parametric approaches using linear architectures, such as radial basis functions (Lagoudakis and Parr, 2003), and nonlinear architectures, such as neural networks (Tesauro, 1992), have been extensively explored. However, this research largely remains within the scope of Samuel's paradigm: most approaches (with notable exceptions discussed below) are based on a fixed parametric architecture, and a parameter estimation method is used to approximate value functions, such as temporal-difference learning (Sutton and Barto, 1998; Tsitsiklis and Van Roy, 1997) or least squares projection (Bradtke and Barto, 1996; Boyan, 1999; Nedic and Bertsekas, 2003; Lagoudakis and Parr, 2003). There has also been significant work on non-parametric methods for approximating value functions, including nearest neighbor methods (Gordon, 1995) and kernel density estimation (Ormoneit and Sen, 2002). Although our approach is also non-parametric, it differs from kernel density estimation and nearest neighbor techniques by extracting a distance measure through modeling the underlying graph or manifold. Non-parametric kernel methods based on Hilbert spaces have also been applied to value function approximationm, including support vector machines (Dietterich and Wang, 2002) and Gaussian processes (Rasmussen and Kuss, 2004). Note that in this approach, the kernel is largely hand-engineered, such as the Gaussian kernel. Our approach can be viewed as extending this work using an automatically generated data-dependent graph or diffusion kernel (Kondor and Vert, 2004).

2.2 Representation Learning

The problem of learning representations has a long history in AI. Amarel (1968) was an early pioneer, advocating the study of representation learning through global state space analysis. Amarel's ideas motivated much subsequent research on representation discovery (Subramanian, 1989; Utgoff and Stracuzzi, 2002), and many methods for discovering global state space properties like "bottlenecks" and "symmetries" have been studied (McGovern, 2002; Ravindran and Barto, 2003; Mannor et al., 2004). However, this past research lacked a formal framework showing how the geometrical analysis of a state space analysis can be transformed into representations for approximating value functions, a hallmark of our approach.

There have been several attempts at overcoming the limitations of traditional function approximators, such as radial basis functions. In particular, it has been recognized that Euclidean smoothing methods do not incorporate geometric constraints intrinsic to the environment: states close in Euclidean distance may be far apart on the manifold. Dayan (1993) proposed the idea of building *successor representations*. While this approach was restricted to policy evaluation in simple discrete MDPs, and did not formally build on manifold or graph-theoretic concepts, the key idea of constructing representations that are faithful to the underlying dynamics of the MDP was a key motivation underlying this work. Drummond (2002) also pointed out the nonlinearities that value functions typically exhibit, and used techniques from computer vision to detect nonlinearities. Neither of these studies formulated the problem of value function approximation as approximating functions on a graph or manifold, and both were restricted to discrete MDPs. There have been several attempts to dynamically allocate basis functions to regions of the state space based on the nonuniform occupancy probability of visiting a region (e.g., (Kretchmar and Anderson, 1999)), but these methods do not construct the basis functions adaptively. Finally, there has also been research on finding common structure among the set of value functions on a given state space, where only the goal location is changed (Foster and Dayan, 2002), assuming a probabilistic generative (mixture) model of a value function, and using maximum likelihood estimation techniques. Proto-value functions can be viewed similarly as the building block of the set of value functions on a given state space, except that they are constructed without the need to make such parametric assumptions.

The proposed approach can be viewed as automatically generating subspaces, on which to project the value function, using spectral analysis of operators on graphs (Chung, 1997). This differs fundamentally from many past attempts at basis function generation, for example tuning a pre-defined set of functions (Menache et al., 2005), dynamically allocating new basis functions based on the non-uniform density of state space trajectories (Kretchmar and Anderson, 1999), or generating task-specific tabular basis functions for factored MDPs using the error in approximating a particular value function (Poupart et al., 2002). Our work also provides a theoretical foundation for recent attempts to automate the learning of task structure in hierarchical reinforcement learning, by discovering "symmetries" or "bottlenecks" (McGovern, 2002; Ravindran and Barto, 2003; Mannor et al., 2004; Simsek and Barto, 2004). Proto-value functions provide a unified framework for studying three problems that face RL systems: geometric structure discovery, representation learning (Dayan, 1993), and finally, actual value function approximation that takes into account smoothness with respect to the intrinsic geometry of the space.

This research also builds on recent work on manifold and spectral learning, including ISOMAP (Tenenbaum et al., 2000), LLE (Roweis and Saul, 2000), Laplacian eigenmaps (Belkin and Niyogi, 2004), and diffusion geometries (Coifman et al., 2005a,b,c). One major difference is that these methods have largely (but not exclusively) been applied to nonlinear dimensionality reduction and semi-supervised learning on graphs (Belkin and Niyogi, 2004; Zhou, 2005; Coifman et al., 2005a), whereas our work focuses on approximating (real-valued) value functions on graphs. Although related to regression on graphs (Niyogi et al., 2003), the problem of value function approximation is fundamentally different: the set of target values is not known a priori, but must be inferred through an iterative process of computing an approximate fixed point of the Bellman backup operator, and projecting these iterates onto subspaces spanned by the basis functions. Furthermore, value function approximation introduces new challenges not present in supervised learning or dimensionality reduction: the set of samples is not specified a priori, but must be collected through *active* exploration of the state space. The techniques we present, such as representation policy iteration (Mahadevan, 2005c), address the unique challenges posed by the problem of value function approximation.

3. Approximation Methods for Solving Markov Decision Processes

We begin by introducing the Markov decision process (MDP) model, and describe methods for approximately solving MDPs. This paper is principally about choosing a basis for approximation of real-valued functions called value functions, which are central to solving Markov decision processes (Puterman, 1994). For much of the past 50 years, previous work has modeled value functions as vectors in a Euclidean space $\mathbb{R}^d$. The fundamental novel idea underlying our approach is that we treat value functions as elements of a *vector space on a graph or manifold*. This difference is fundamental: it enables constructing basis functions that capture the large-scale (irregular) topology of the graph, and approximating (value) functions on these graphs by projecting them onto the space spanned by these bases.

3.1 Markov Decision Processes

A discrete Markov decision process (MDP) $M = (S, A, P^a_{ss'}, R^a_{ss'})$ is defined by a finite set of discrete states S, a finite set of actions A, a transition model $P^a_{ss'}$ specifying the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R^a_{ss'}$ specifying a scalar cost or reward (Puterman, 1994). In Section 9, we will also investigate continuous Markov decision processes, where the set of states $\subseteq \mathbb{R}^d$. Abstractly, a value function is a mapping $S \rightarrow \mathbb{R}$ or equivalently (in discrete MDPs) a vector $\in \mathbb{R}^{|S|}$. Given a policy $\pi : S \rightarrow A$ mapping states to actions, its corresponding value function V^π specifies the expected long-term discounted sum of rewards received by the agent in any given state s when actions are chosen using the policy. Any optimal policy π^* defines the same unique optimal value function V^* which satisfies the nonlinear constraints

$$V^*(s) = \max_a \left(R_{sa} + \gamma \sum_{s' \in S} P^a_{ss'} V^*(s') \right)$$

where $R_{sa} = \sum_{s' \in s} P^a_{ss'} R^a_{ss'}$ is the expected immediate reward. The expected long-term discounted sum of rewards at each state is called a value function, defined by a fixed (deterministic) policy π as

$$V^{\pi}(s) = R_{s\pi(s)} + \gamma \sum_{s' \in S} P_{ss'}^{\pi(s)} V^{\pi}(s') \tag{1}$$

We can compactly represent the above equation as

$$V^{\pi} = T^{\pi}(V^{\pi})$$

where the *Bellman operator* T^{π} on the space of value function can be written as

$$T^{\pi}(V) = R_{s\pi(s)} + \gamma \sum_{s'} P_{ss'}^{\pi(s)} V(s')$$

This formulation shows that the value function associated with a policy is the fixed point of the (linear) operator T^{π}. The optimal value function can be similarly written as the fixed point of the (nonlinear) operator T^*, where

$$T^{*}(V) = \max_a \left(R_{sa} + \gamma \sum_{s'} P_{ss'}^{a} V(s') \right)$$

The *action value* $Q^*(s,a)$ represents a convenient reformulation of the value function, where the long-term value of performing a first, and then acting optimally according to V^*, is defined as

$$Q^*(s,a) = E\left(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a \right) \tag{2}$$

where r_{t+1} is the actual reward received at the next time step, and s_{t+1} is the state resulting from executing action a in state s_t. The (optimal) action value formulation provides an alternative way to write the Bellman equation:

$$Q^{*}(s,a) = R_{sa} + \gamma \sum_{s'} P_{ss'}^{a} \max_{a'} Q^*(s', a')$$

3.2 Dynamic Programming

The optimal value function V^* can be computed using *value iteration* and *policy iteration* (Puterman, 1994), which can be viewed as finding the fixed point of the Bellman operator T^* using the Euclidean unit vector orthonormal basis $(\phi_1, \ldots, \phi_{|S|})$ for the space $\mathbb{R}^{|S|}$, where the unit vector $\phi_i = [0 \ldots 1 \ldots 0]$ has a 1 only in the i^{th} position. Value iteration can be briefly summarized as

$$V_{n+1} = T^* V_n$$

where V_n is the approximated value function at stage n. This algorithm is provably convergent since the operator T^* is a contraction (in some weighted $\mathbb{L}^{\infty}$ norm (Puterman, 1994)), hence successive iterates converge to the optimal value function V^*. *Policy iteration* consists of interleaving two phases: policy evaluation and (greedy) policy improvement. Policy evaluation can be viewed as finding the fixed point

$$V^{\pi_k} = T^k V^{\pi_k}$$

by solving the resulting system of $|S|$ linear equations, where π_k is the policy chosen at the k^{th} stage. Given the resulting value function V^{π_k}, the policy improvement phase finds a one-step "greedy" improvement of this policy π_{k+1}, where

$$\pi_{k+1} \in \max_a \left(R_{sa} + \gamma \sum_{s'} P^a_{ss'} V^{\pi_k} \right)$$

Policy iteration is guaranteed to converge in a finite number of steps because every policy improvement step results in a different ("better") policy, unless the policy being evaluated is already optimal, and the space of policies is finite, albeit in general very large. Both these methods assume that the transition model $P^a_{ss'}$ and reward model $R^a_{ss'}$ are known.

3.3 Approximate Dynamic Programming

Approximate dynamic programming methods combine machine learning (or statistical) methods with these exact methods. For example, approximate value iteration can be described as selecting at each stage k a sample of states $S^k = (s^k_1, \ldots, s^k_m)$, computing an approximate backed up value $\hat{T}$ on the sample S^k, and then using some parametric or non-parametric regression method to approximately represent V^{k+1}. Methods for approximating value functions can be abstractly viewed as projecting a value function onto a subspace of the vector space $\mathbb{R}^{|S|}$. The subspace can be defined in many ways, for example, using CMAC, polynomial or radial basis functions.

We mentioned above that the Bellman backup operators T^π and T^* are *contractions* in $\mathbb{R}^{|S|}$, with respect to a weighted $\mathbb{L}^\infty$-norm (Puterman, 1994). Bounds for the error in the greedy policy based on the error in approximating a given value function exist (Bertsekas and Tsitsiklis, 1996). Unfortunately, these bounds are not as helpful in designing function approximators, which are usually designed to minimize a quadratic $\mathbb{L}^2$ norm. Recent work (Munos, 2005, 2003) has addressed this discrepancy by deriving bounds on the error of approximation with respect to the $\mathbb{L}^2$ norm. Also, there has been been research on designing approximation methods that are projections in $\mathbb{L}^\infty$ norm (Guestrin et al., 2001).

Let us define a set of basis functions $\Phi = \{\phi_1, \ldots, \phi_k\}$, where each basis function $\phi_i : S \to \mathbb{R}$. The basis function matrix Φ is an $|S| \times k$ matrix, where each column is a particular basis function evaluated over the state space, and each row is the set of all possible basis functions evaluated on a particular state. Approximating a value function using the matrix Φ can be viewed as projecting the value function onto the column space spanned by the basis functions ϕ_i,

$$V^\pi \approx \Phi w^\pi = \sum_i w^\pi_i \phi_i$$

where the weight vector w needs to be determined. The basis function matrix can easily be extended to action value functions, where $\phi_i(s,a) : S \times A \to \mathbb{R}$, giving

$$Q^\pi(s,a) \approx \sum_i w^\pi_i \phi_i(s,a)$$

Combining this linear representation of a value function with Equation 1 above gives us

$$\sum_i w^\pi_i \phi_i(s) = R_{s\pi(s)} + \gamma \sum_{s'} P^{\pi(s)}_{ss'} \sum_i w^\pi_i \phi_i(s) \tag{3}$$

However, in general it is no longer guaranteed that this equation has a fixed point solution, since the "projected" Bellman operator on the right-hand side of (3) may no longer be a contraction. There are two approaches to approximately solving this modified equation: the *Bellman residual* and the *least-squares fixpoint* methods (Munos, 2003; Lagoudakis and Parr, 2003).

Both of these arise from a natural desire to explore a minimum-norm *least-squares* solution to the overdetermined system of equations in Equation 3. Let us first rewrite Equation 3 in matrix form, making it explicit that we are seeking an approximate solution:

$$\Phi w^\pi \approx R + \gamma P^\pi \Phi w^\pi$$

where P^π is the transition matrix defined by the policy π. The Bellman residual method minimizes the length of the difference between the original value function V^π and the backed up function $T^\pi(V^\pi)$.

$$\begin{aligned} V^\pi &= R^\pi + \gamma P^\pi V^\pi \\ \Phi w^\pi &\approx R^\pi + \gamma P^\pi \Phi w^\pi \\ (\Phi - \gamma P^\pi \Phi)\, w^\pi &\approx R^\pi \end{aligned}$$

Now, a straightforward least-squares approach leads us to the following equation for the weight vector:

$$w^\pi = \left((\Phi - \gamma P^\pi \Phi)^T (\Phi - \gamma P^\pi \Phi)\right)^{-1} (\Phi - \gamma P^\pi \Phi)^T R^\pi$$

In contrast, the Bellman least-squares fixpoint approach minimizes the length of the projected residual in the columnspace of the basis Φ. To ensure that the vector $T^\pi(V^\pi)$ defined on the right-hand side is in the column space of Φ, we can project it onto the subspace defined by the basis functions Φ. Since the orthogonal projection onto this subspace is given by $(\Phi^T\Phi)^{-1}\Phi^T$ (whose rows one may interpret as the basis dual to Φ):

$$\begin{aligned} w^\pi &= (\Phi^T\Phi)^{-1}\Phi^T \left[R + \gamma P^\pi \Phi w^\pi\right] \\ (\Phi^T\Phi) w^\pi &= \Phi^T \left[R + \gamma P^\pi \Phi w^\pi\right] \\ (\Phi^T\Phi) w^\pi - \gamma \Phi^T P^\pi \Phi w^\pi &= \Phi^T R \\ \Phi^T (\Phi - \gamma P^\pi \Phi)\, w^\pi &= \Phi^T R \\ w^\pi &= \left(\Phi^T(\Phi - \gamma P^\pi \Phi)\right)^{-1} \Phi^T R \end{aligned}$$

The last equation is solvable except possibly for a finite number of values of γ (Koller and Parr, 2000). It is also straightforward to modify the above derivation to take into account the non-uniform probability of visiting each state $\rho^\pi(s)$, giving a weighted least-squares solution.

$$w^\pi = \left(\Phi^T D_\rho^\pi (\Phi - \gamma P^\pi \Phi)\right)^{-1} \Phi^T D_\rho^\pi R \tag{4}$$

where D_ρ^π is a diagonal matrix with entries $\rho^\pi(s)$ reflecting the possible nonuniform distribution of frequencies of state visitations, which can be used to selectively control the approximation error.

So far, we have assumed that the model of the system P^π to be controlled using the policy π is known, as is the payoff or reward function R. When this knowledge is unavailable, a variety of sampling methods can be devised, such as computing low-rank approximations of the matrix to be inverted based on an initial random walk. In Section 6, we will describe the least-squares policy iteration (LSPI) method developed by Lagoudakis and Parr (2003).

The question of whether the Bellman residual method or the least-squares fixpoint method is superior has been discussed, but not resolved in previous work (Munos, 2003; Lagoudakis and Parr, 2003). In particular, when a model of the system is unknown, the least-squares fixpoint method has the advantage of being able to use a single sample of experience (s, a, s', r), instead of requiring two independent samples. For the most part, this paper will focus on the least-squares fixpoint. However, in Section 7.2, we will discover an interesting case where the Bellman residual method works significantly better.

We have also assumed thus far that the set of basis functions ϕ_i is designed by hand for each specific MDP. We now turn to describing a general framework that enables automatically computing a set of basis functions called proto-value functions, and then demonstrate how these can be combined with least-squares methods such as LSPI.

4. Proto-Value Functions: A Brief Overview

In many applications the state space S is naturally embedded in $\mathbb{R}^d$ as a sub-manifold, or some other type of geometrically structured subset. Often one constructs basis functions on S by considering basis functions defined on $\mathbb{R}^d$ (e.g. Gaussian bells) and restricting them to S. This approach has the advantage that the basis functions are easily designed and are defined on all of $\mathbb{R}^d$. However they are not adapted to the geometry of S, and often the difficulty of picking good basis functions is a consequence of their non-adaptiveness to S. Also, if the goal is to approximate value functions, it seems natural to assume that these functions are *smooth with respect to the natural geometry of the state space*. In general, functions that are smooth when viewed intrinsically as functions on the state space may not be smooth when viewed in the ambient space, and conversely. Our goal is to automatically construct basis functions which are intrinsic to the state space, adapted to its geometry, and guaranteed to provide good approximation of functions that are intrinsically smooth.

We now introduce the concept of proto-value functions, generated by spectral analysis of a graph using the graph Laplacian (Chung, 1997). We show some simple MDPs where proto-value functions are effective for low-dimensional approximation of value functions. For simplicity, we assume here that both the graph and the value function to be approximated are known. In subsequent sections, we relax these assumptions and address the more challenging problem of learning the graph and computing the fixed point of the Bellman operator to approximate value functions.

4.1 Basis Functions from the Graph Laplacian

Proto-value functions are a set of geometrically customized basis functions for approximating value functions, which result from conceptualizing value functions not as vectors in a general Euclidean space $\mathbb{R}^d$, but on a manifold (or more general classes of sets) whose geometry can be modeled using a (undirected or directed) graph G. The advantage of this approach is that the set of basis functions can be automatically determined by spectral analysis of random walks on the graph. The inherent smoothness of value functions arises from the Bellman equations: qualitatively, the (action) value at a given state (or state action pair) is a linear function of the corresponding (action) values at neighboring states [1]. Examples of value functions with these properties are shown in Figure 7.

Proto-value functions approximate value functions based on construction of basis functions that are derived from the topology of the state space. The set of all functions on a graph forms a Hilbert space, that is a complete vector space endowed with an inner product. Spectral analysis of operators on the graph yields an orthonormal set of basis functions for approximating any function on a graph. The notion of producing bases by diagonalizing self-adjoint (or symmetric) *operators* is standard practice in approximation theory (Deutsch, 2001). While many operators can be defined on a graph, in this paper proto-value functions will be defined using the *graph Laplacian* (Chung, 1997). Basis functions are constructed by spectral analysis of the graph Laplacian, a self-adjoint (or symmetric) operator on the space of functions on the graph, related closely to the random walk operator.

For simplicity, assume the underlying state space is represented as an undirected unweighted graph $G = (V, E)$ (the more general case of weighted graphs is analyzed in Section 5, and directed graphs are discussed in Section 10.3). The *combinatorial Laplacian* L is defined as the operator

$$L = D - A \ ,$$

where D is a diagonal matrix called the *valency* matrix whose entries are row sums of the adjacency matrix A. The combinatorial Laplacian L acts on any given function $f : S \rightarrow \mathbb{R}$, mapping vertices of the graph (or states) to real numbers.

$$Lf(x) = \sum_{y \sim x} (f(x) - f(y))$$

for all y adjacent to x. Consider the chain graph G shown in Figure 2 consisting of a set of vertices linked in a path of length N. Given any function f on the chain graph, the combinatorial Laplacian can be viewed as a discrete analog of the well-known Laplace partial differential equation

$$\begin{aligned} Lf(v_i) &= (f(v_i) - f(v_{i-1})) + (f(v_i) - f(v_{i+1})) \\ &= (f(v_i) - f(v_{i-1})) - (f(v_{i+1}) - v_i) \\ &= \nabla f(v_i, v_{i-1}) - \nabla f(v_{i+1}, f(v_i)) \\ &= \Delta f(v_i) \end{aligned}$$

Functions that solve the equation $\Delta f = 0$ are called *harmonic functions* (Axler et al., 2001) For example the "saddle" function $x^2 - y^2$ is harmonic on $\mathbb{R}^2$. Eigenfunctions of Δ are functions $f \neq 0$ such that $\Delta f = \lambda f$, where λ is an eigenvalue of Δ. If the domain is the unit circle S^1, the trigonometric functions $\sin(k\theta)$ and $\cos(k\theta)$, for any $k \in \mathbb{Z}$ are eigenfunctions, associated with *Fourier* analysis.

The spectral analysis of the Laplace operator on a graph consists in finding the solutions (eigenvalues and eigenfunctions) of the equation

$$L\phi_\lambda = \lambda\phi_\lambda \ ,$$

1. More quantitatively, for classes of continuous MDPs associated with "diffusion-like" processes, the Bellman operator is a smoothing operator because of elliptic estimates.

where L is the combinatorial Laplacian, ϕ_λ is an eigenfunction associated with eigenvalue λ. The term "eigenfunction" is used to denote what is traditionally referred to as an "eigenvector" in linear algebra, because it is natural to view the Laplacian eigenvectors as functions that map each vertex of the graph to a real number.

A fundamental property of the graph Laplacian is that projections of functions on the eigenspace of the Laplacian produce globally the smoothest approximation, which respects the underlying manifold. More precisely, it is easy to show that

$$\langle f, Lf \rangle = \sum_{u \sim v} w_{uv} (f(u) - f(v))^2$$

where this so-called *Dirichlet sum* is over the edges $u \sim v$ of the graph G [2], and w_{uv} denotes the weight on the edge. From the standpoint of regularization, this property is crucial since it implies that rather than smoothing using properties of the ambient Euclidean space, smoothing takes the underlying manifold (graph) into account.

Graphs are fundamentally coordinate-free objects. The eigenfunctions of the graph Laplacian provide a coordinate system, that is they embed the vertices of a graph onto $\mathbb{R}^k$. Typically, the embedding is done using the smoothest k eigenfunctions correspond to the smallest eigenvalues of the Laplacian. The Laplacian has unique spectral properties that makes these embeddings reflect spatial regularities, unlike other embeddings derived from a spectral analysis of the adjacency matrix (see Figure 1). We can define an optimization problem of finding a basis function ϕ which embeds vertices of a discrete state space S onto the real line $\mathbb{R}$, i.e. $\phi : S \to \mathbb{R}$, such that neighboring vertices on the graph are generally mapped to neighboring points on the real line:

$$\min_{\phi} \sum_{u \sim v} (\phi(u) - \phi(v))^2 \, w_{uv}$$

In order for this minimization problem to be well-defined, and to eliminate trivial solutions like mapping all graph vertices to a single point, e.g. $\phi(u) = 0$, we can impose an additional "length" normalization constraint of the form:

$$\langle \phi, \phi \rangle_G = \phi^T D \phi = 1$$

where the valency matrix D measures the importance of each vertex. The solution to this minimization problem can be shown to involve finding the smallest (non-zero) eigenvalue of the generalized eigenvalue problem (Belkin and Niyogi, 2004; Ng et al., 2001a; Lafon, 2004).

$$L\phi = \lambda D \phi$$

where ϕ is the eigenvector providing the desired embedding. More generally, the smoothest eigenvectors associated with the k smallest eigenvalues form the desired proto-value function basis. If the graph is connected, then D is invertible, and the above generalized eigenvalue problem can be converted into a regular eigenvalue problem

$$D^{-1} L \phi = \lambda \phi$$

2. Here, $u \sim v$ and $v \sim u$ denote the same edge, and are summed over only once.

where $D^{-1}L$ is the discrete Laplacian operator. Note that this operator is closely related to the natural random walk defined on an undirected graph, since $D^{-1}L = I - P$, where

$$P = D^{-1}A\ .$$

While the random walk operator P is not symmetric unless D is a multiple of the identity (i.e. the graph is regular), its eigenvalues are related to those of symmetric operator, namely the *normalized* Laplacian

$$\mathcal{L} = I - D^{\frac{1}{2}} P D^{-\frac{1}{2}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

whose spectral analysis is highly revealing of the large-scale structure of a graph (Chung, 1997). We will describe detailed experiments below showing the effectiveness of proto-value functions constructed from spectral analysis of the normalized Laplacian.

In Section 8.1 we will see that Laplacian eigenfunctions can be compactly represented on interesting structured graphs obtained by natural graph operations combining smaller graphs.

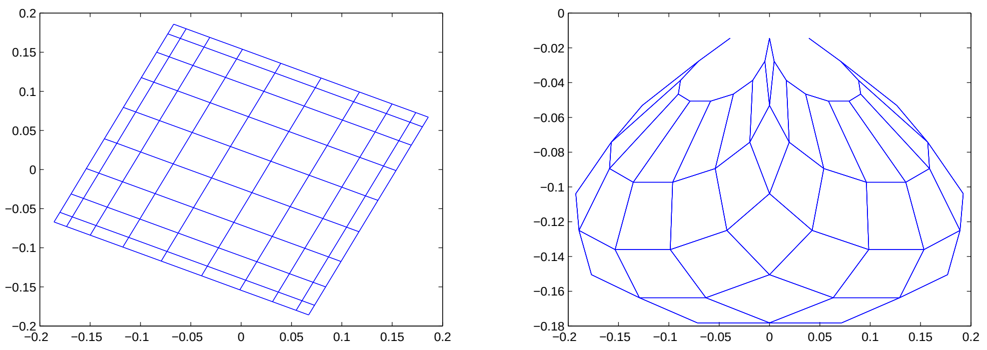


Figure 1: The eigenfunctions of the Laplacian provide a coordinate-system to an initially coordinate-free graph. This figure shows an embedding in $\mathbb{R}^2$ of a 10×10 grid world environment using "low-frequency" (smoothest) eigenvectors of a graph operator. The plot on the left shows an embedding using the eigenvectors associated with the second and third lowest eigenvalues of the graph Laplacian. The plot on the right shows the embedding using the smoothest eigenvectors associated with the first and second highest eigenvalues of the adjacency matrix as coordinates. The spatial structure of the grid is preserved nicely under the Laplacian embedding, but appears distorted under the adjacency spectrum. Section 8 explains the regularity of Laplacian embeddings of structured spaces such as grids, hypercubes, and tori.

To summarize, proto-value functions are abstract Fourier basis functions that represent an orthonormal basis set for approximating any value function. Unlike trigonometric Fourier basis functions, proto-value functions or Laplacian eigenfunctions are learned from the graph topology. Consequently, they capture large-scale geometric constraints, and examples of proto-value functions showing this property are shown below.

4.2 Examples of Proto-Value Functions

This section illustrates proto-value functions, showing their effectiveness in approximating a given value function. For simplicity, we confine our attention to discrete deterministic environments which an agent is assumed to have completely explored, constructing an undirected graph representing the accessibility relation between adjacent states through single-step (reversible) actions. Later, we will extend these ideas to larger non-symmetric discrete and continuous domains, and where the full control learning problem of finding an optimal policy will be addressed. Note that topological learning does not require estimating probabilistic transition dynamics of actions, since representations are learned in an off-policy manner by spectral analysis of a random walk diffusion operator (the combinatorial or normalized Laplacian).

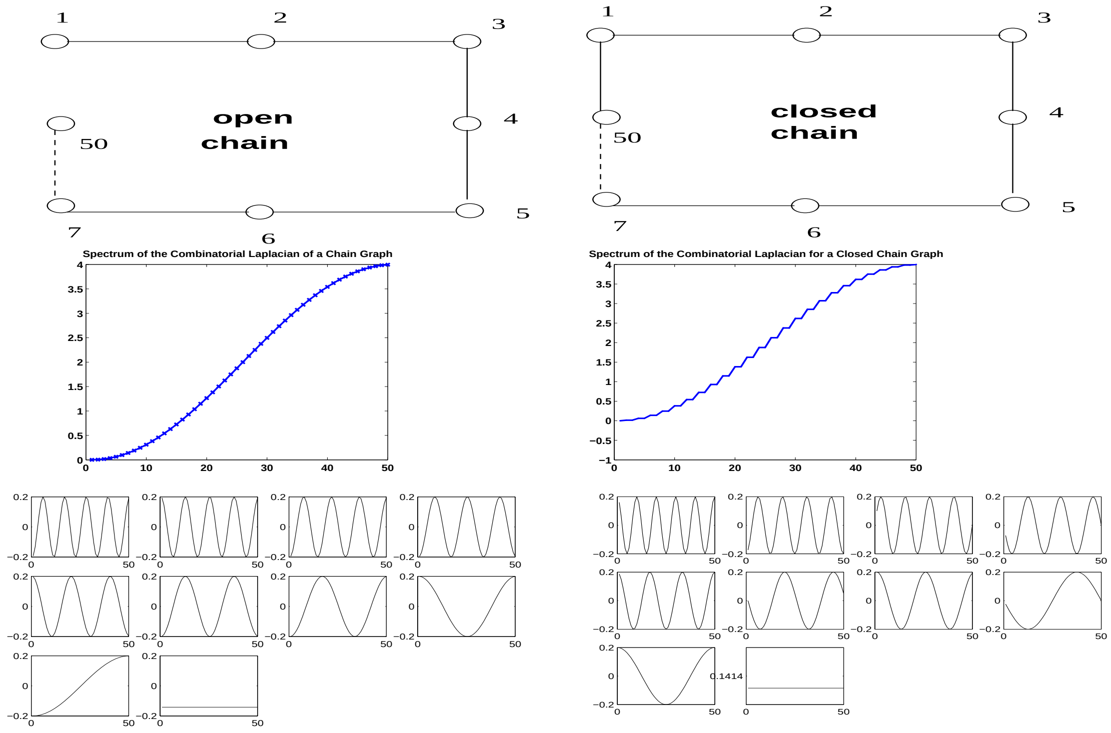


Figure 2: Proto-value functions for a 50 state open and closed chain graph, computed as the eigenfunctions of the combinatorial graph Laplacian. Low-order eigenfunctions are smoother than higher-order eigenfunctions. Note the differences in the spectrum: the closed chain appears more jagged since many eigenvalues have multiplicity two. Each proto-value function is actually a vector $\in \mathbb{R}^{50}$, but depicted as a continuous function for clarity.

The chain MDP, originally studied in (Koller and Parr, 2000), is a sequential open chain of varying number of states, where there are two actions for moving left or right along the chain. The reward structure can vary, such as rewarding the agent for visiting the middle states, or the end states. Instead of using a fixed state action encoding, our approach automatically derives a customized encoding that reflects the *topology* of the chain. Figure 2 shows the proto-value functions that are created for an open and closed chain using the combinatorial Laplacian.

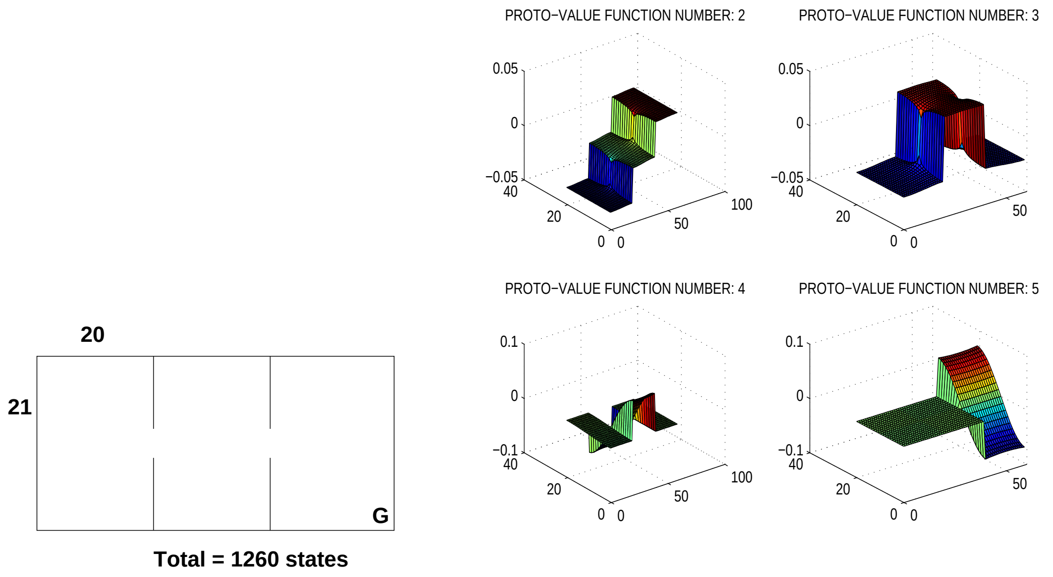


Figure 3: The low-order eigenfunctions of the combinatorial Laplace operator for a three room deterministic grid world environment. The numbers indicate the size of each room. The horizontal axes in the plots represent the length and width of the multiroom environment.

Figure 3 shows proto-value functions automatically constructed from an undirected graph of a three room deterministic grid world. These basis functions capture the intrinsic smoothness constraints that value functions on this environment must also abide by. Proto-value functions are effective bases because of these constraints.

Proto-value functions extend naturally to continuous Markov decision processes as well. Figure 4 illustrates a graph connecting nearby states in a 2-dimensional continuous state space for the inverted pendulum task, representing the angle of the pole and the angular velocity. The resulting graph on 500 states with around $15,000$ edges is then analyzed using spectral graph theoretic methods, and a proto-value function representing the second eigenfunction of the graph Laplacian is shown. One challenge in continuous state spaces is how to interpolate the eigenfunctions defined on sample states to novel states. We describe the *Nyström* method in Section 9 that enables out-of-sample extensions and interpolates the values of proto-value functions from sampled points to novel points. More refined techniques based on comparing harmonic analysis on the set with harmonic analysis in the ambient space are described in (Coifman et al., 2005b; Coifman and Maggioni, 2004). Figure 5 shows a graph of $\approx 7,000$ edges on 670 states generated in the mountain car domain. The construction of these graphs depends on many issues, such as a sampling method that selects a subset of the overall states visited to construct the graph, the local metric used to connect states by edges, and so on. The details of graph construction will be discussed in Section 9.4.

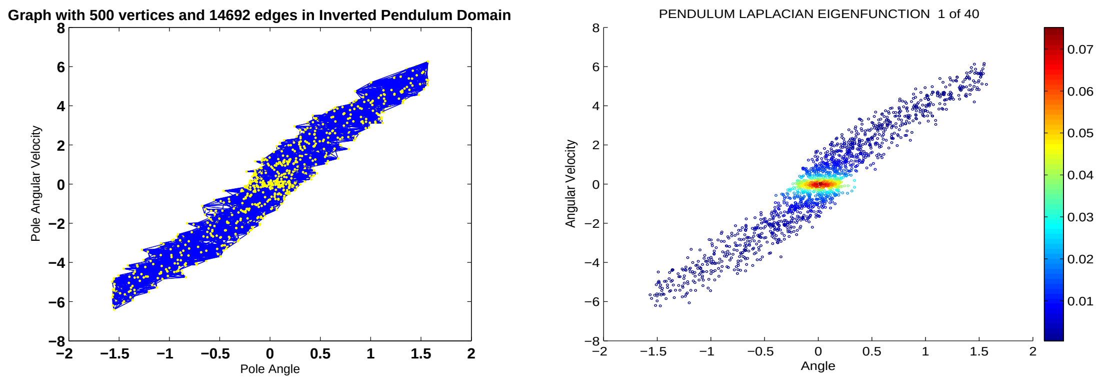


Figure 4: Proto-value functions can also be learned for continuous MDPs by constructing a graph on a set of sampled states. Shown here from left to right are a graph of 500 vertices and $\approx 15,000$ edges, along with a proto-value function (the second eigenfunction of the graph Laplacian) for the inverted pendulum task generated from a random walk. See also Figure 23 and Figure 25.

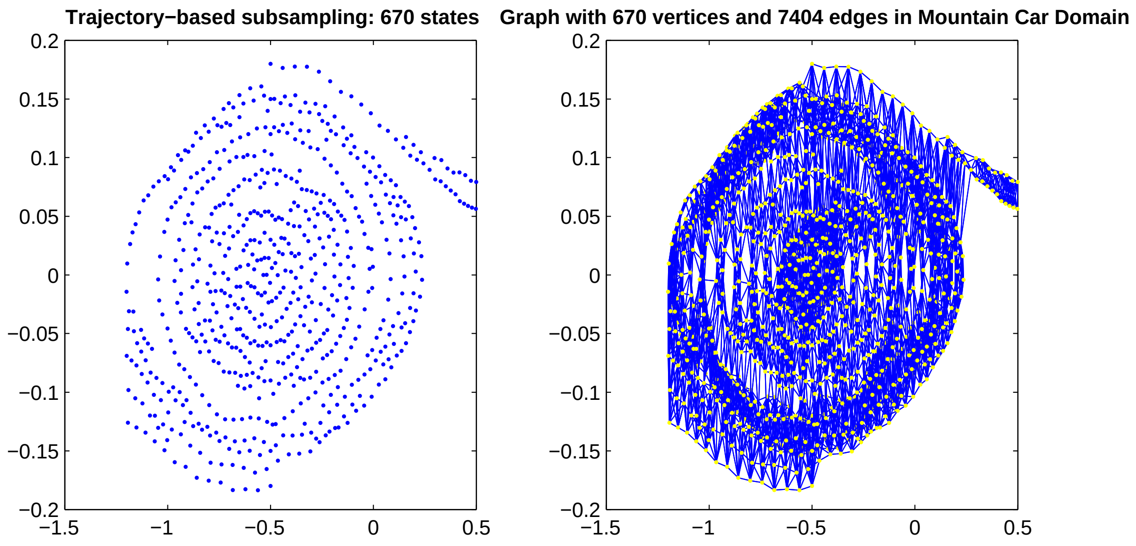


Figure 5: A graph of 670 vertices and 7,404 edges generated from a random walk in the mountain car domain.

```
Laplacian Algorithm $(G, k, \mathcal{O}, V^\pi)$:

// $G$: Weighted undirected graph of $|V| = n$ nodes
// $k$: Number of basis functions to be used
// $V^\pi$: Target value function, specified on the vertices of graph $G$

    1. Compute the operator $\mathcal{O}$, which can be any among the combinatorial graph Laplacian, the
       normalized graph Laplacian, the random walk operator etc. (these and other choices of graph
       operators are discussed in detail in Section 5).
    2. Compute the proto-value basis functions as the $k$ smoothest eigenvectors of $\mathcal{O}$ on the graph
       $G$, and collect them as columns of the basis function matrix $\Phi$, a $|\mathcal{S}| \times k$ matrix.[a] The basis
       function embedding of a state $v$ is the encoding of the corresponding vertex $\phi(v)$ given by the
       $v^{th}$ row of the basis function matrix. Let $\tilde{\phi}_i$ be the $i$-th column of $\tilde{\Phi} = ((\Phi^*\Phi)^{-1}\Phi^*)^*$, the
       dual basis of $\Phi$ ($\tilde{\Phi}^*\Phi = I$).
    3. Linear Least-Squares Projection: In the linear least-squares approach, the approximated
       function $\hat{V}^\pi$ is specified as
           $$\hat{V}^\pi = \sum_{i=1}^{k} \langle V^\pi, \tilde{\phi}_i \rangle \phi_i \quad (5)$$
    4. Nonlinear Least-Squares Projection: In the nonlinear least-squares approach (Mallat, 1998),
       the approximated function $\hat{V}^\pi$ is specified as
           $$\hat{V}^\pi = \sum_{i \in I_k(V^\pi)} \langle V^\pi, \tilde{\phi}_i \rangle \phi_i \quad (6)$$
       where $I_k(V^\pi)$ is the set of indices of the $k$ basis functions with the largest inner product
       (in absolute value) with $V^\pi$. Hence in nonlinear approximation, the selected basis functions
       depend on the function $V^\pi$ to be approximated.

a. Note that depending on the graph operator being used, the smoothest eigenvectors may either be the
   ones associated with the smallest or largest eigenvalues.
```

Figure 6: Pseudo-code of linear and nonlinear least-squares algorithms for approximating a known function on a graph G by projecting the function onto the eigenspace of the graph Laplacian.

4.3 Least-Squares Approximation using Proto-Value Functions

Figure 6 describes linear and nonlinear least-squares algorithms that approximate a given target (value) function on a graph by projecting it onto the eigenspace of the graph Laplacian. In Section 6, we present a more general algorithm that constructs the graph from a set of samples, and also approximates the (unknown) optimal value function. For now, this simplified situation is sufficient to illustrate the unique properties of proto-value functions.

Figure 7 shows the results of linear least-squares for the three-room environment. The agent is only given a goal reward of $R = 10$ for reaching the absorbing goal state marked

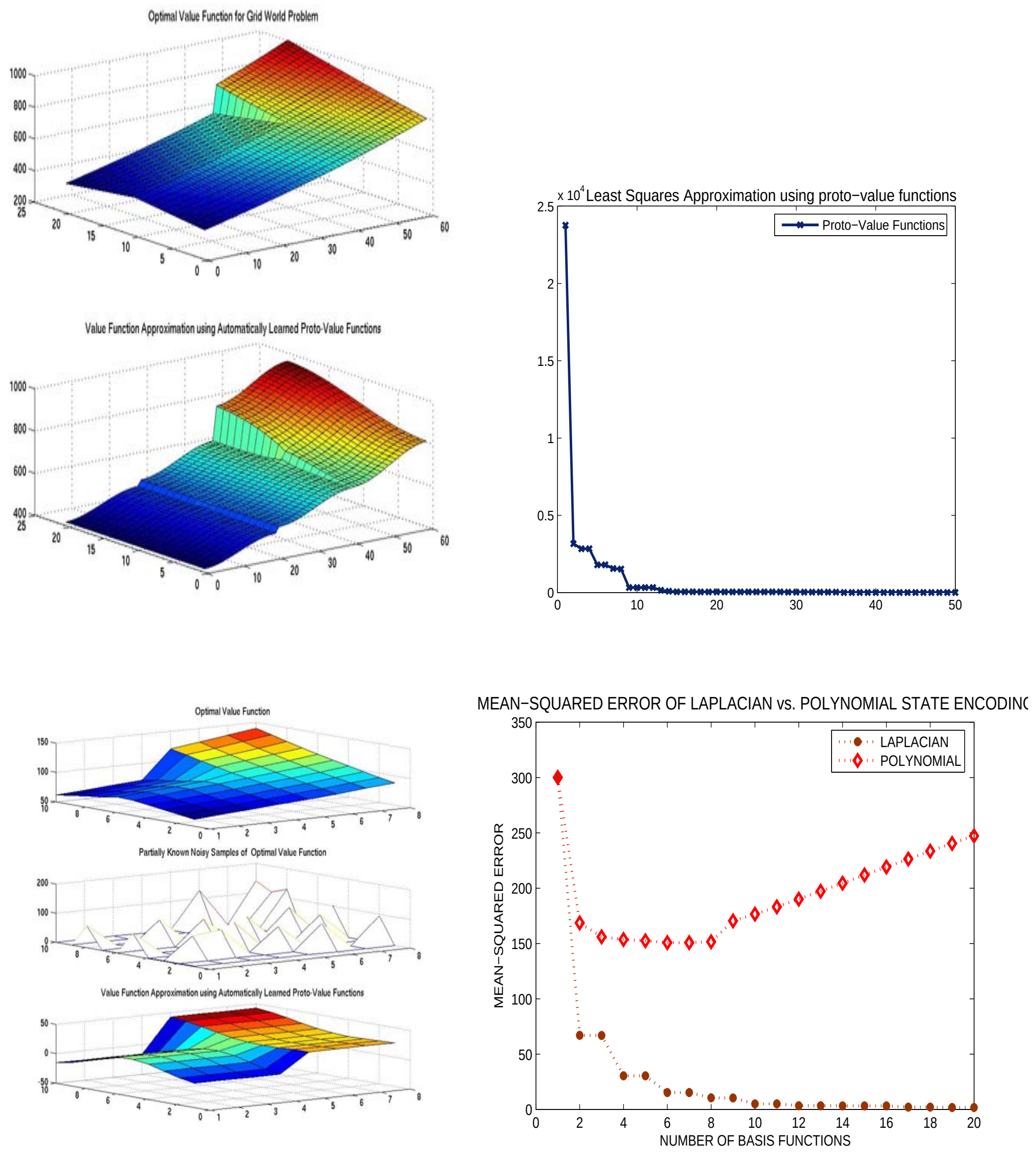


Figure 7: Top Left: the optimal value function for a three-room grid world MDP (top plot) is a vector $\in \mathbb{R}^{1260}$, but is nicely approximated by a linear least-squares approximation (bottom plot) onto the subspace spanned by the smoothest 20 proto-value functions. Top Right: mean-squared error in approximating the optimal three-room MDP value function. Bottom left: proto-value function approximation (bottom plot) using 5 basis functions from a noisy partial (18%) set of samples (middle plot) from the optimal value function for a 80 state two-room grid world (top plot), simulating an early stage in the process of policy learning. Bottom right: mean squared error in value function approximation for a square 20×20 grid world using proto-value functions (bottom curve) versus handcoded polynomial basis functions (top curve).

G in Figure 3. The discount factor γ is set to 0.99. Although value functions for the three-room environment are high dimensional objects in $\mathbb{R}^{1260}$, a reasonable likeness of the optimal value function is achieved using only 20 proto-value functions.

Figure 7 also plots the error in approximating the value function as the number of proto-value functions is increased. With 20 basis functions, the high-dimensional value function vector is fairly accurately reconstructed. To simulate value function approximation under more realistic conditions based on partial noisy samples, we generated a set of noisy samples, and compared the approximated function with the optimal value function. Figure 7 also shows the results for a two-room grid world of 80 states, where noisy samples were filled in for about 18% of the states. Each noisy sample was produced by scaling the exact value by Gaussian noise with mean $\mu = 1$ and variance $\sigma^2 = 0.1$. As the figure shows, the distinct character of the optimal value function is captured even with very few noisy samples.

Finally, Figure 7 shows that proto-value functions improve on a polynomial basis studied in (Koller and Parr, 2000; Lagoudakis and Parr, 2003). In this scheme, a state s is mapped to $\phi(s) = [1, s, s^2, \ldots, s^i]^T$ where $i \ll |S|$. Notice that this mapping does not preserve neighbors; each coordinate is not a smooth function on the state space, and will not represent efficiently smooth functions on the state space. The figure compares the least mean square error with respect to the optimal (correct) value function for both the handcoded polynomial encoding and the automatically generated proto-value functions for a square grid world of size 20×20. There is a dramatic reduction in error using the learned Laplacian proto-value functions compared to the handcoded polynomial approximator. Notice how the error using polynomial approximation gets worse at higher degrees. Mathematically this is impossible, however computationally instability can occur because the least-squares problem with polynomials of high degree becomes extremely ill-conditioned. The same behavior manifests itself below in the control learning experiments (see Table 2).

To conclude this section, Figure 8 compares the linear and nonlinear least-squares techniques in approximating a value function on a two-room discrete MDP, with a reward of 50 in state 420 and a reward of 25 in state 1. The policy used here is a random walk on an undirected state space graph, where the edges represent states that are adjacent to each other under a single (reversible) action (north, south, east, or west). Note that both approaches take significantly longer to produce a good approximation since the value function is quite non-smooth, in the sense it has a large gradient. In the second paper (Maggioni and Mahadevan, 2006), we show that a diffusion wavelet basis performs significantly better at approximating functions that are non-smooth in local regions, but smooth elsewhere. Since Laplacian eigenfunctions use Fourier-style global basis functions, they produce global "ripples" as shown when reconstructing such piecewise smooth functions. Notice also that linear methods are significantly worse than nonlinear methods, which choose basis functions adaptively based on the function being approximated. However, given that the value function being approximated is not initially known, we will largely focus on linear least-squares methods in the first paper. Other approximation methods could be used, such as projection pursuit methods (Mallat, 1998), or best basis methods (Coifman et al., 1993; Bremer et al., 2004).

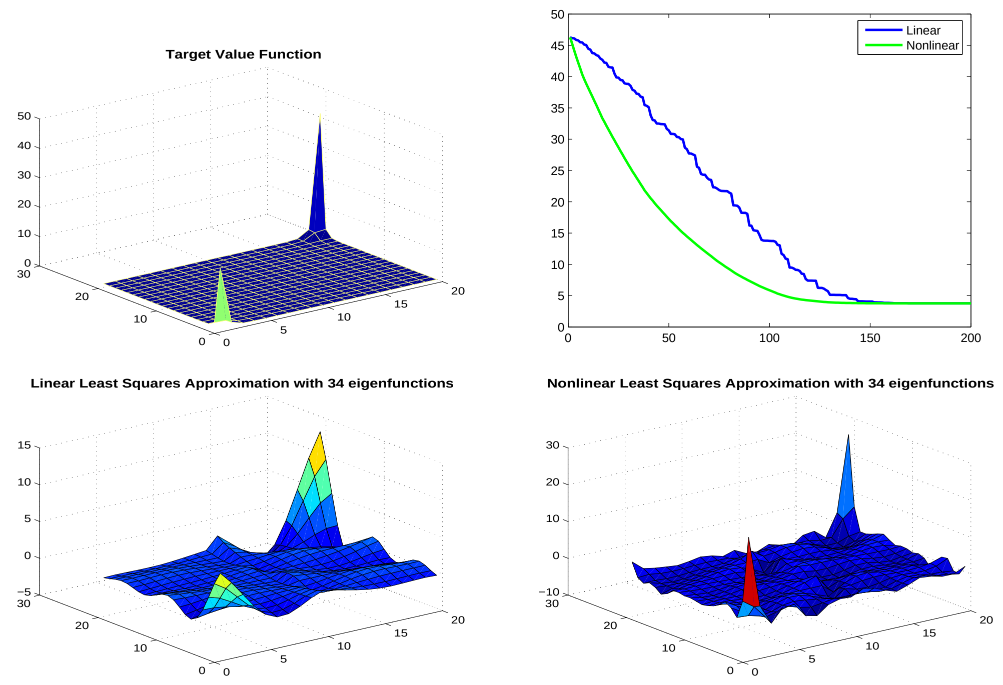


Figure 8: This figure compares linear vs. nonlinear least squares approximation of the target value function shown in the first plot on the left on a two room grid world MDP, where each room is of size 21×10. The environment has two rewards, one of 50 in state $|S| = 21 \times 10 \times 2 = 420$ and the other of size 20 in state 1. The target value function represents the value function associated with a random walk on an unweighted undirected graph connecting states that are adjacent to each other under a single (reversible) action. The two plots on the bottom show the reconstruction of the value function with 34 proto-value functions using linear and nonlinear least-squares methods. The graph on the top right plots the reconstruction error, showing that the nonlinear least-squares approach is significantly quicker in this problem.

5. Technical Background

In this section, we provide a brief overview of the mathematics underlying proto-value functions, in particular, spectral graph theory (Chung, 1997) and its continuous counterpart, analysis on Riemannian manifolds (Lee, 2003).

5.1 Riemannian Manifolds

This section introduces the Laplace-Beltrami operator in the general setting of Riemannian manifolds (Rosenberg, 1997), as a prelude to describing the Laplace-Beltrami operator in the more familiar setting of graphs (Chung, 1997). Riemannian manifolds have been actively studied recently in machine learning in several contexts. It has been known for over 50 years that the space of probability distributions forms a Riemannian manifold, with the Fisher information metric representing the Riemann metric on the tangent space. This observation has been applied to design new types of kernels for supervised machine learning (Lafferty and Lebanon, 2005) and faster policy gradient methods using the natural Riemannian gradient on a space of parametric policies (Kakade, 2002; Bagnell and Schneider, 2003; Peters et al., 2003). In recent work on manifold learning, Belkin and Niyogi (2004) have studied semi-supervised learning in Riemannian manifolds, where a large set of *unlabeled* points are used to extract a representation of the underlying manifold and improve classification accuracy. The Laplacian on Riemannian manifolds and its eigenfunctions (Rosenberg, 1997), which form an orthonormal basis for square-integrable functions on the manifold (Hodge's theorem), generalize Fourier analysis to manifolds. Historically, manifolds have been applied to many problems in AI, for example configuration space planning in robotics, but these problems assume a model of the manifold is known (Latombe, 1991; Lavalle, 2005), unlike here where only samples of a manifold are given. Recently, there has been rapidly growing interest in manifold learning methods, including ISOMAP (Tenenbaum et al., 2000), LLE (Roweis and Saul, 2000), and Laplacian eigenmaps (Belkin and Niyogi, 2004). These methods have been applied to nonlinear dimensionality reduction as well as *semi-supervised* learning on graphs (Belkin and Niyogi, 2004; Zhou, 2005; Coifman et al., 2005a).

We refer the reader to Rosenberg (1997) for an introduction to Riemannian geometry and properties of the Laplacian on Riemannian manifolds. Let $(\mathcal{M}, g)$ be a smooth compact connected Riemannian manifold. The Laplacian is defined as

$$\Delta = \operatorname{div}\operatorname{grad} = \frac{1}{\sqrt{\det\ g}} \sum_{ij} \partial_i \left(\sqrt{\det\ g}\ \ g^{ij} \partial_j \right)$$

where div and grad are the Riemannian divergence and gradient operators. We say that $\phi : \mathcal{M} \to \mathbb{R}$ is an eigenfunction of Δ if $\phi \neq 0$ and there exists $\lambda \in \mathbb{R}$ such that

$$\Delta\phi = \lambda\phi\,.$$

If $\mathcal{M}$ has a boundary, special conditions need to be imposed. Typical boundary conditions include Dirichlet conditions, enforcing $\phi = 0$ on $\partial\mathcal{M}$ and Neumann conditions, enforcing $\partial_\nu \phi = 0$, where ν is the normal to $\partial\mathcal{M}$. The set of λ's for which there exists an eigenfunction is called the spectrum of Δ, and is denoted by $\sigma(\Delta)$. We always consider eigenfunctions which have been $\mathbb{L}^2$-normalized, i.e. $||\phi||_{\mathbb{L}^2(\mathcal{M})} = 1$.

The quadratic form associated to the Laplacian is the Dirichlet integral

$$S(f) := \int_{\mathcal{M}} ||\mathrm{grad} f||^2 \mathrm{d}\,\mathrm{vol} = \int_{\mathcal{M}} f \Delta f \mathrm{d}\,\mathrm{vol} =< \Delta f, f >_{\mathcal{L}^2(\mathcal{M})} = ||\mathrm{grad} f||_{\mathbb{L}^2(\mathcal{M})}$$

where $\mathbb{L}^2(\mathcal{M})$ is the space of square integrable functions on $\mathcal{M}$, with respect to the natural Riemannian volume measure. It is natural to consider the space of functions $\mathcal{H}^1(\mathcal{M})$ defined as follows:

$$\mathcal{H}^1(\mathcal{M}) = \left\{ f \in \mathbb{L}^2(\mathcal{M}) : ||f||_{\mathcal{H}^1(\mathcal{M})} := ||f||_{\mathbb{L}^2(\mathcal{M})} + S(f) \right\}\,. \quad (7)$$

So clearly $\mathcal{H}^1(\mathcal{M}) \subsetneq \mathbb{L}^2(\mathcal{M})$ since functions in $\mathcal{H}^1(\mathcal{M})$ have a square-integrable gradient. The smaller the $\mathcal{H}^1$-norm of a function, the "smoother" the function is, since it needs to have small gradient. Observe that if ϕ_λ is an eigenfunction of Δ with eigenvalue λ, then $S(\phi_\lambda) = \lambda$: the larger is λ, the larger the square-norm of the gradient of the corresponding eigenfunction, i.e. the more oscillating the eigenfunction is.

Theorem 1 *(Hodge (Rosenberg, 1997)): Let $(\mathcal{M}, g)$ be a smooth compact connected oriented Riemannian manifold. The spectrum $0 \leq \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_k \leq \ldots$, $\lambda_k \to +\infty$, of Δ is discrete, and the corresponding eigenfunctions $\{\phi_k\}_{k \geq 0}$ form an orthonormal basis for $\mathbb{L}^2(\mathcal{M})$.*

In particular any function $f \in \mathbb{L}^2(\mathcal{M})$ can be expressed as $f(x) = \sum_{k=0}^{\infty} \langle f, \phi_k \rangle \phi_k(x)$, with convergence in $\mathbb{L}^2(\mathcal{M})$.

5.2 Spectral Graph Theory

Operator	**Definition**	**Spectrum**
Adjacency	A	$\lambda \in \mathbb{R}$, $\lvert\lambda\rvert \leq \max_v d_v$
Combinatorial Laplacian	$L = D - A$	Positive semi-definite, $\lambda \in [0, 2\max_v d_v]$
Normalized Laplacian	$\mathcal{L} = I - D^{-1/2} A D^{-1/2}$	Positive semi-definite, $\lambda \in [0, 2]$
Random Walk	$T = D^{-1} A$	$\lambda \in [-1, 1]$

Table 1: Some operators on undirected graphs.

Many of the key ideas from the continuous manifold setting translate into the (simpler) discrete setting studied in spectral graph theory, which is increasingly finding more applications in AI, from image segmentation (Shi and Malik, 2000) to clustering (Ng et al., 2002). The Laplace-Beltrami operator now becomes the graph Laplacian (Chung, 1997; Cvetkovic et al., 1980), from which an orthonormal set of basis functions $\phi_1^G(s), \ldots, \phi_k^G(s)$ can be extracted. The graph Laplacian can be defined in several ways, such as the *combinatorial* Laplacian and the *normalized* Laplacian, in a range of models from undirected graphs with $(0, 1)$ edge weights to directed arbitrary weighted graphs with loops (Chung, 1997).

Generally speaking, let $G = (V, E, W)$ denote a weighted undirected graph with vertices V, edges E and weights w_{ij} on edge $(i, j) \in E$. The degree of a vertex v is denoted as d_v. The adjacency matrix A can be viewed as a binary weight matrix W (we use A and W interchangeably below). A few operators of interest on graphs are listed in Table 1. D in the table denotes the *valency* matrix, or a diagonal matrix whose entries correspond to the degree of each vertex (i.e, the row sums of A). If the graph is undirected, all eigenvalues are real, and the spectral theorem implies that the adjacency (or weight) matrix can be diagonalized to yield a complete set of orthonormal basis functions:

$$A = V \Lambda V^T$$

where V is an orthonormal set of eigenvectors that span the Hilbert space of functions on the graph $H_G : V \to \mathbb{R}$, and Λ is a diagonal matrix whose entries are the eigenvalues of A. It is helpful to make explicit the effect of applying each operator to any function f on the graph, so for the adjacency matrix A, we have:

$$Af(u) = \sum_{v \sim u} f(v) w_{uv}$$

Here, f is a function mapping each vertex of the graph to a real number, and $f(v)$ is the value of the function on vertex v. The adjacency operator *sums* the values of the function at neighboring vertices, where $u \sim v$ denotes an edge between u and v.

The graph Laplacians are a discrete version of the Laplacian on a Riemannian manifold (Rosenberg, 1997). The convergence of the discrete Laplacian to the continuous Laplacian on the underlying manifold under uniform sampling conditions has been shown recently in (Belkin and Niyogi, 2005). The *combinatorial Laplacian* $L = D - A$ acts on a function f as

$$Lf(u) = \sum_{v \sim u} (f(u) - f(v))\, w_{uv}$$

Unlike the adjacency matrix operator, the combinatorial Laplacian acts as a *difference* operator. More importantly, the Laplacians yield a *positive-semidefinite* matrix. It is easy to show that for any function f, the quadratic form $f^T Lf \geq 0$. More precisely:

$$\langle f, Lf \rangle = f^T Lf = \sum_{u \sim v \in E} (f(u) - f(v))^2 w_{uv}$$

where the sum is over all edges in E. In the context of regression, this property shows that the Laplacian produces an approximation that takes the edges of the graph into account, and smoothing respects the manifold. This property is crucial for value function approximation. Another striking property of both graph Laplacians is that the constant function $f = \mathbf{1}$ is an eigenvector associated with the eigenvalue $\lambda_0 = 0$, which is crucial for proper embedding of structured graphs like grids (as was shown earlier in Figure 1).

The random walk operator on a graph, given by $D^{-1}W$, is not symmetric, but it is spectrally similar to the symmetric normalized graph Laplacian operator. The *normalized Laplacian* $\mathcal{L}$ of the graph G is defined as $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ or in more detail

$$\mathcal{L}(u,v) = \begin{cases} 1 - \frac{w_{vv}}{d_v} & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{w_{uv}}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{L}$ is a symmetric self-adjoint operator, and its spectrum (eigenvalues) lie in the interval $\lambda \in [0, 2]$ (this is a direct application of Cauchy-Schwartz inequality of $\langle \mathcal{L}f, f \rangle$). Furthermore, the eigenvector associated with $\lambda_0 = 0$ is the constant eigenvector. For a general graph G, $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Thus, $D^{-1}A = D^{-\frac{1}{2}}(I - \mathcal{L})D^{\frac{1}{2}}$. That is, the random walk operator $D^{-1}A$ is similar to $I - \mathcal{L}$, so both have the same eigenvalues, and the eigenvectors of the random walk operator are the eigenvectors of $I - \mathcal{L}$ pointwise multiplied by $D^{-\frac{1}{2}}$. The normalized Laplacian $\mathcal{L}$ acts on functions by

$$\mathcal{L}f(u) = \frac{1}{\sqrt{d_u}} \sum_{v \sim u} \left(\frac{f(u)}{\sqrt{d_u}} - \frac{f(v)}{\sqrt{d_v}} \right) w_{uv}\,. \tag{8}$$

The spectrum of the graph Laplacian has an intimate relationship to global properties of the graph, such as volume, “dimension”, bottlenecks and mixing times of random walks. The latter are connected with the first non-zero eigenvalue λ_1, often called the *Fiedler value* (Fiedler, 1973). The lower the Fiedler value, the easier it is to partition the graph into components without breaking too many edges.

5.3 Function Approximation on Graphs and Manifolds

We now discuss the approximation of functions using inner product spaces generated by a graph. The $\mathcal{L}^2$ norm of a function on graph G is

$$||f||_2^2 = \sum_{x \in G} |f(x)|^2 d(x) \ .$$

The gradient of a function on graph G is

$$\nabla f(i,j) = w(i,j)(f(i) - f(j)) \ ,$$

if there is an edge e connecting i to j, 0 otherwise. The smoothness of a function on a graph, can be measured by the *Sobolev norm*

$$||f||_{\mathcal{H}^1}^2 = ||f||_2^2 + ||\nabla f||_2^2 = \sum_x |f(x)|^2 d(x) + \sum_{x \sim y} |f(x) - f(y)|^2 w(x,y) \, . \qquad (9)$$

The first term in this norm controls the size (in terms of $\mathcal{L}^2$-norm) for the function f, and the second term controls the size of the gradient. The smaller $||f||_{\mathcal{H}^1}$, the smoother is f. We will assume that the value functions we consider have small $\mathcal{H}^1$ norms, except at a few points, where the gradient may be large. Important variations exist, corresponding to different measures on the vertices and edges of G.

Given a set of basis functions ϕ_i that span a subspace of an inner product space, for a fixed precision ϵ, a value function V^π can be approximated as

$$\left\| V^\pi - \sum_{i \in S(\epsilon)} \alpha_i^\pi \phi_i \right\| \leq \epsilon$$

with $\alpha_i = \langle V^\pi, \phi_i \rangle$ since the ϕ_i's are orthonormal, and the approximation is measured in some norm, such as $\mathcal{L}^2$ or $\mathcal{H}^2$. The goal is to obtain representations in which the index set $S(\epsilon)$ in the summation is as small as possible, for a given approximation error ϵ. This hope is well founded at least when V^π is smooth or piecewise smooth, since in this case it should be compressible in some well chosen basis $\{e_i\}$.

The *normalized* Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is related to the above notion of smoothness since

$$\langle f, \mathcal{L} f \rangle = \sum_x f(x) \, Lf(x) = \sum_{x,y} w(x,y)(f(x) - f(y))^2 = ||\nabla f||_2^2 \ ,$$

which should be compared with (9).

The eigenfunctions of the Laplacian can be viewed as an orthonormal basis of global Fourier smooth functions that can be used for approximating any value function on a graph. The projection of a function f on S onto the top k eigenvectors of the Laplacian is the smoothest approximation to f with k vectors, in the sense of the norm in $\mathcal{H}^1$. A potential drawback of Laplacian approximation is that it detects only global smoothness, and may poorly approximate a function which is not globally smooth but only piecewise smooth, or with different smoothness in different regions. These drawbacks are addressed by diffusion wavelets (Coifman and Maggioni, 2004), and in fact partly motivated their construction.

6. Algorithmic Details

In this section, we begin the detailed algorithmic analysis of the application of proto-value functions to solve Markov decision processes. We introduce a novel variant of a least-square policy iteration method called representation policy iteration (RPI) (Mahadevan, 2005c). We will analyze three variants of RPI, beginning with the most basic version in this section, and then describing two extensions of RPI to continuous and factored state spaces in Section 8 and Section 9.

6.1 Least-Squares Approximation from Samples

The basics of least-squares approximation of value functions were reviewed in Section 3.3. As discussed below, the weighted least-squares approximation given by Equation 4 can be estimated from samples. This sampling approach eliminates the necessity of knowing the state transition matrix P^π and reward function R, but it does introduce another potential source of error. We modify the earlier description to approximating action-value or $Q^\pi(s,a)$ functions, which are more convenient to work with for the purposes of learning. LSPI (Lagoudakis and Parr, 2003) and other similar approximation methods (Bradtke and Barto, 1996) approximate the true action-value function $Q^\pi(s,a)$ for a policy π using a set of (handcoded) basis functions $\phi(s,a)$ that can be viewed as doing dimensionality reduction: the true action value function $Q^\pi(s,a)$ is a vector in a high dimensional space $\mathbb{R}^{|S|\times|A|}$, and using the basis functions amounts to reducing the dimension to $\mathbb{R}^k$ where $k \ll |S| \times |A|$. The approximated action value is thus

$$\hat{Q}^\pi(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j$$

where the w_j are weights or parameters that can be determined using a least-squares method. Let Q^π be a real (column) vector $\in \mathbb{R}^{|S|\times|A|}$. The column vector $\phi(s,a)$ is a real vector of size k where each entry corresponds to the basis function $\phi_j(s,a)$ evaluated at the state action pair (s,a). The approximate action-value function can be written as $\hat{Q}^\pi = \Phi w^\pi$, where w^π is a real column vector of length k and Φ is a real matrix with $|S| \times |A|$ rows and k columns. Each row of Φ specifies all the basis functions for a particular state action pair (s,a), and each column represents the value of a particular basis function over all state action pairs. The least-squares fixed-point approximation $T_\pi Q^\pi \approx Q^\pi$, where T_π is the Bellman backup operator, yields the following solution for the coefficients:

$$\begin{aligned} A w^\pi &= b \\ \left(\Phi^T D^\pi_\rho (\Phi - \gamma P^\pi \Phi)\right) w^\pi &= \Phi^T D^\pi_\rho R \end{aligned}$$

A and b can be estimated from a database of transitions collected from some source, e.g. a random walk, by noticing that both A^π and b^π can be written as the sum of many rank one outer product multiplications:

$$\begin{aligned} A &= \sum_{s \in A} \sum_{a \in A} \phi(s,a)\rho^\pi(s,a) \left(\phi(s,a) - \gamma \sum_{s \in S} P^a_{ss'} \phi(s', \pi(s')) \right)^T \\ &= \sum_{s \in A} \sum_{a \in A} \rho^\pi(s,a) \sum_{s' \in S} P^a_{ss'} \left[\phi(s,a) \left(\phi(s,a) - \gamma \phi(s', \pi(s')) \right)^T \right] \end{aligned}$$

where $\rho^\pi(s,a)$ is the probability of executing action pair (s,a) under policy π. Similarly, b can be written as

$$\begin{aligned} b &= \Phi^T D^\pi_\rho R \\ &= \sum_{s \in S} \sum_{a \in A} \rho^\pi(s,a) \sum_{s' \in S} P^a_{ss'} \left[\phi(s,a) R^a_{ss'} \right] \end{aligned}$$

The A matrix and b vector can be estimated as the sum of many rank-one matrix summations from a database of stored samples.

$$\begin{aligned} \tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t, a_t) \left(\phi(s_t, a_t) - \gamma \phi(s'_t, \pi(s'_t)) \right)^T \\ \tilde{b}^{t+1} &= \tilde{b}^t + \phi(s_t, a_t) r_t \end{aligned}$$

where (s_t, a_t, r_t, s'_t) is the t^{th} sample of experience from a trajectory generated by the agent (using some random or guided policy), and $\tilde{A}^0$ and $\tilde{b}^0$ are set to 0.

In contrast, the *Bellman residual* approach requires two independent samples of each transition from s_t to s'_t and s''_t, resulting in the modified update equation:

$$\tilde{A}^{t+1} = \tilde{A}^t + \left(\phi(s_t, a_t) - \gamma \phi(s'_t, \pi(s'_t)) \right) \left(\phi(s_t, a_t) - \gamma \phi(s''_t, \pi(s''_t)) \right)^T \quad (10)$$

$$\tilde{b}^{t+1} = \tilde{b}^t + \left(\phi(s_t, a_t) - \gamma \phi(s'_t, \pi(s'_t)) \right) r_t \quad (11)$$

A "naive" implementation of the residual approach is to reuse the same next state sample twice, by setting $s'_t = s''_t$.[3] In either case, a modified policy iteration procedure can now be defined, which starts with a policy π defined by an initial weight vector w, and then repeatedly invoking LSTDQ to find the updated weights w', and terminating when the difference $\|w - w'\| \leq \epsilon$.

6.2 Representation Policy Iteration

With this brief overview of least-squares policy iteration, we can now introduce the Representation Policy Iteration (RPI) method, which adds a representation learning step that automatically computes the basis functions Φ which are assumed to be hand engineered in LSPI and previous linear approximation methods. We present the complete algorithm in Figure 9 in the context of LSPI, although clearly other parameter estimation methods such as TD(λ) (Sutton and Barto, 1998) can easily be modified to use proto-value functions as well.

3. The LSPI MATLAB code, a modified version of which was used in our experiments, implements this naive version.

As Figure 9 shows, RPI can be broadly construed as consisting of two phases, which can be interleaved (although that is not explicitly shown). In the representation learning phase, an initial random walk (perhaps guided by an informed policy) is carried out to obtain samples of the underlying manifold on the state space. The number of samples needed is an empirical question which will be investigated in further detail in Section 8 and Section 9. Given this set of samples, an undirected graph is constructed in one of several ways: two states can be connected by a unit cost edge if they represent temporally successive states; alternatively, a local distance measure such as k-nearest neighbor can be used to connect states, which is particularly useful in the experiments on continuous domain reported in Section 9.5. From the graph, proto-value functions are computed using one of the spectral operators, for example the combinatorial or normalized Laplacian. There are several ways of normalizing the graph Laplacians, which will be described later. The smoothest eigenvectors of the graph Laplacian (that is, associated with the smallest eigenvalues) are used to form the suite of proto-value functions. The number of proto-value functions needed is again an empirical question that will be explored in the experiments. The encoding $\phi(s) : S \rightarrow \mathbb{R}^k$ of a state is computed as the value of the k proto-value functions on that state. To compute a state action encoding, a number of alternative strategies can be followed: the figure shows the most straightforward method of simply replicating the length of the state encoding by the number of actions and setting all the vector components to 0 except those associated with the current action. More sophisticated schemes are possible (and necessary for continuous actions), and are being investigated currently.

The control learning phase consists of simply using LSPI (or some similar least-squares or TD method) with the proto-value functions. The least-squares approach was described in detail in the previous section. An optional basis adaptation step can be used to eliminate proto-value functions whose coefficients fall below a desired threshold. More sophisticated basis adaptation methods are possible as well.

7. Experimental Results: Small Discrete MDPs

We illustrate the RPI algorithm in this section using simple discrete MDPs, and also compare the performance of proto-value functions against traditional function approximators, such as polynomials and radial basis functions. Results on larger domains, including continuous and factored MDPs, are described in the following sections.

7.1 Chain Domain

Figure 10 and Figure 11 show the results of running the Representation Policy Iteration (RPI) algorithm on a 50 node chain graph, following the display format used by Lagoudakis and Parr (2003). The agent is rewarded in states 10 and 41 by +1, and the reward is 0 otherwise. The optimal policy is to go right in states 1 through 9 and 26 through 41 and left in states 11 through 25 and 42 through 50. The number of samples initially collected was set at $10,000$. The discount factor was set at $\gamma = 0.8$. By increasing the number of desired basis functions, it is possible to get very accurate approximation, although as Figure 11 shows, even a crude approximation using 5 basis functions is sufficient to learn a close to optimal policy. Using 20 basis functions, the learned policy is exact. The basis adaption parameter was set to $\delta = 0$, so all proto-value functions were retained on each step.

```
RPI Algorithm $(T, N, \epsilon, P, \mathcal{O}, \delta)$:

// $T$: Number of initial random walk trials
// $N$: Maximum length of each trial
// $\epsilon$ : Convergence condition for policy iteration
// $P$: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used
// $\delta$: Parameter for basis adaptation

                    Representation Learning Phase

  1. Perform a random walk of $T$ trials, each of maximum $N$ steps, and store the states visited in $S$.
  2. Build an undirected weighted graph $G$ from $S$, in one of the ways described in the text. For
     example, connect state $i$ to state $j$ if it is one of its $k$ "nearest" neighbors (using some norm
     on the state space, such as Manhattan or Euclidean distance), assiging a weight $w(i,j) =
     e^{-(\frac{\|s_i - s_j\|}{\delta})^2}$ Alternatively, connect state $i$ to state $j$ with a weight of 1 if the pair $(i,j)$ form
     temporally successive states $\in S$.
  3. Construct one of the operators $\mathcal{O}$ on graph $G$ as discussed in the text. Compute the $k$
     smoothest eigenvectors of $\mathcal{O}$ on the graph $G$, and collect them as columns of the basis function
     matrix $\Phi$, a $|S| \times k$ matrix. The encoding of a state action pair $(s,a)$ is given as $e_a \otimes \phi(s)$,
     where $e_a$ is the unit vector corresponding to action $a$, $\phi(s)$ is the $s^{th}$ row of $\Phi$, and $\otimes$ is the
     tensor product.

                    Control Learning Phase

  1. Initialize $w^0 \in \mathbb{R}^k$ to a random vector.
  2. Repeat the following steps:
     (a) Set $i \leftarrow i+1$. For each transition $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}$, compute low rank approximations
         of matrix $A$ and $b$ as follows:

             $\tilde{A}^{t+1} = \tilde{A}^t + \phi(s_t, a_t)\left(\phi(s_t, a_t) - \gamma\phi(s'_t, a'_t)\right)^T$
             $\tilde{b}^{t+1} = \tilde{b}^t + \phi(s_t, a_t) r_t$

     (b) Solve the system $\tilde{A} w^i = \tilde{b}$
     (c) Optional basis adaptation step: Modify the suite of proto-value functions $\Phi$ by
         discarding those whose coefficients are smaller than $\delta$.
  3. until $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
  4. Return $\hat{Q}^* \approx \sum_i w^i \Phi$ as the approximation to the optimal value function.
```

Figure 9: Pseudo-code of the representation policy iteration (RPI) algorithm for discrete Markov decision processes. For simplicity, the control learning component uses the least-squares policy iteration (LSPI) fixpoint method. Other choices for control learning can be easily incorporated, including the Bellman residual method and TD(λ).

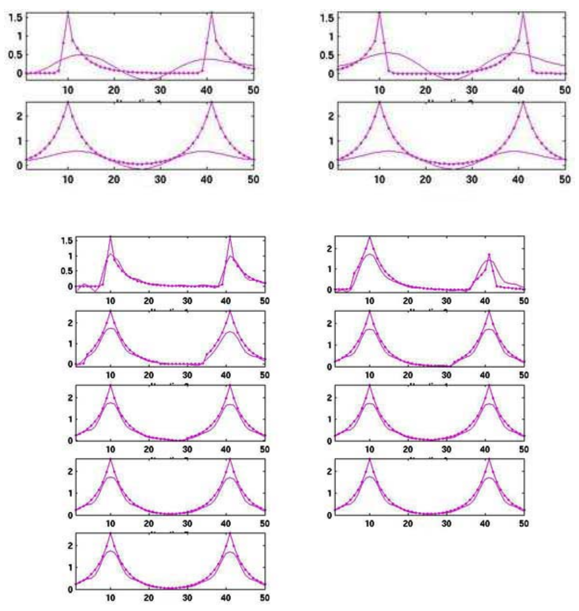


Figure 10: Representation Policy Iteration on a 50 node chain graph, for $k = 5$ basis functions (top four panels) and $k = 20$ (bottom nine panels). Each group of plots shows the state value function for each iteration (in row major order) over the 50 states, where the solid curve is the approximation and the dotted lines specify the exact function. Notice how the value function approximation gets much better at $k = 20$. Although the approximation is relatively poor at $k = 5$, the policy learned turns out to be close to optimal.

Next, we provide a detailed control learning experiment comparing RPI with proto-value functions against two handcoded basis functions, polynomial encoding and radial-basis functions (RBF) (see Table 2). Each row reflects the performance of either RPI using learned basis functions or LSPI with a parametric handcoded basis function (values in parentheses indicate the number of basis functions used for each architecture). Each result is the average of five experiments on a sample of $10,000$ transitions. The two numbers reported are average steps to convergence and the average error in the learned policy (number of incorrect actions). The results show the automatically learned Laplacian basis functions in RPI provide a more stable performance at both the low end (5 basis functions) and at the higher end with $k = 25$ basis functions. As the number of basis functions are increased, RPI takes longer to converge, but learns a more accurate policy. LSPI with RBF is unstable at the low end, converging to a very poor policy for 6 basis functions. LSPI with a degree 5 polynomial approximator works reasonably well, but its performance noticeably degrades at higher degrees, converging to a very poor policy in one step for $k = 15$ and $k = 25$.

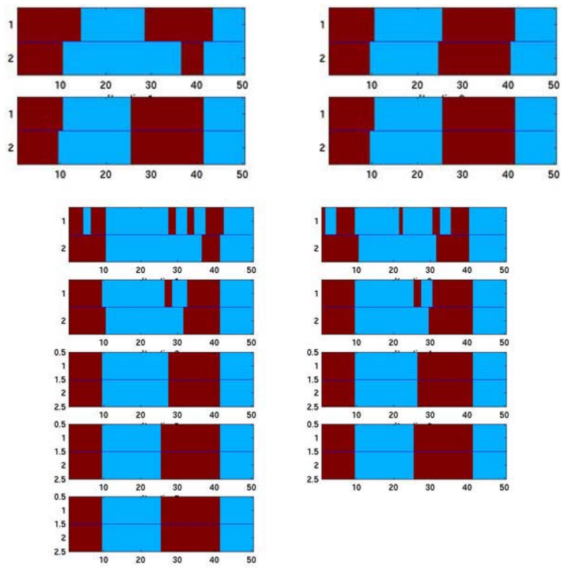


Figure 11: The policies learned at each iteration using Representation Policy Iteration on a 50 node chain graph, for 5 basis functions (top four panels), and 20 basis functions (bottom nine panels) in row major order. Even using 5 basis functions results in a close to optimal policy. The light (blue) color denotes the left action and the dark (red) denotes going right. The top half of each plot is exact, and the bottom is approximate. The display format follows that used by Lagoudakis and Parr (2003).

7.2 Two-Dimensional Grid with Obstacles

One striking property of proto-value functions is their ability to reflect nonlinearities arising from "bottlenecks" in the state space. It has long been recognized that such nonlinearities are the bane of traditional function approximators, and various attempts have been made to try to "fix" their shortcomings (Dayan, 1993; Drummond, 2002). Since proto-value functions reflect large-scale nonlinearities, they automatically reflect bottlenecks like "walls" or "obstacles". Figure 12 contrasts the value function approximation produced by RPI using Laplacian eigenfunctions with that produced by polynomial and RBF function approximators, which yields an approximation that is "blind" to the nonlinearities produced by the walls in the two room grid world MDP.

Interestingly, this domain highlighted significant differences between the Bellman residual approach (which is the one illustrated in Figure 12) versus the least-squares fixpoint method. The fixpoint approach consistently seemed more unstable in this environment, while the residual approach (even when implemented "naively" by reusing the same next state sample s'_t twice) yielded superior results. This finding is somewhat in contrast to that observed by Lagoudakis and Parr (2003), where the least-squares fixpoint method was found to be consistently superior in the chain domain. One possible explanation may be

Method	#Trials	Error
RPI (5)	4.2	-3.8
RPI (15)	7.2	-3
RPI (25)	9.4	-2
LSPI RBF (6)	3.8	-20.8
LSPI RBF (14)	4.4	-2.8
LSPI RBF (26)	6.4	-2.8
LSPI Poly (5)	4.2	-4
LSPI Poly (15)	1	-34.4
LSPI Poly (25)	1	-36

Table 2: This table compares the performance of RPI using automatically learned basis functions with LSPI combined with two handcoded basis functions on a 50 state chain graph problem. See text for explanation.

the global nature of the Laplacian bases. Further work is necessary to explore the impact of basis functions on Bellman residual vs. least-squares fixpoint methods.

Note also that while the approximated value function produced by the Laplacian bases seems much closer to the exact value function than either of the parametric approaches, there are still significant errors introduced in the approximation (e.g., near the walls). These error in value function approximation of course leads to errors in the policy. We explore these tradeoffs in the experiments on larger domains in Section 8 and Section 9, where we directly compare the quality of the learned policy produced by Laplacian bases with that produced by other parametric architectures.

8. Scaling Proto-Value Functions to Large Discrete Markov Decision Processes

Thus far, we have restricted our discussion of proto-value functions to discrete MDPs, showing that they can efficiently represent nonlinear value functions in simple discrete domains more reliably than traditional parametric approximators. In this and the next section, we explore the issue of scaling the Laplacian framework to large discrete and continuous domains. Computing and storing proto-value functions in large discrete domains can be intractable: spectral analysis of the state space graph or diagonalization of the policy transition matrix can be an infeasible eigenvector computation in large domains, even if the matrices are inherently sparse matrices. We describe a general framework for scaling proto-value functions to large factored discrete spaces using properties of product spaces, such as grids, cylinders, and tori.

8.1 Product Spaces: Complex Graphs from Simple Ones

Figure 13 illustrates one general way of scaling proto-value functions to complex structured domains. Building on the theory of graph spectra (Cvetkovic et al., 1980), a hierarchical

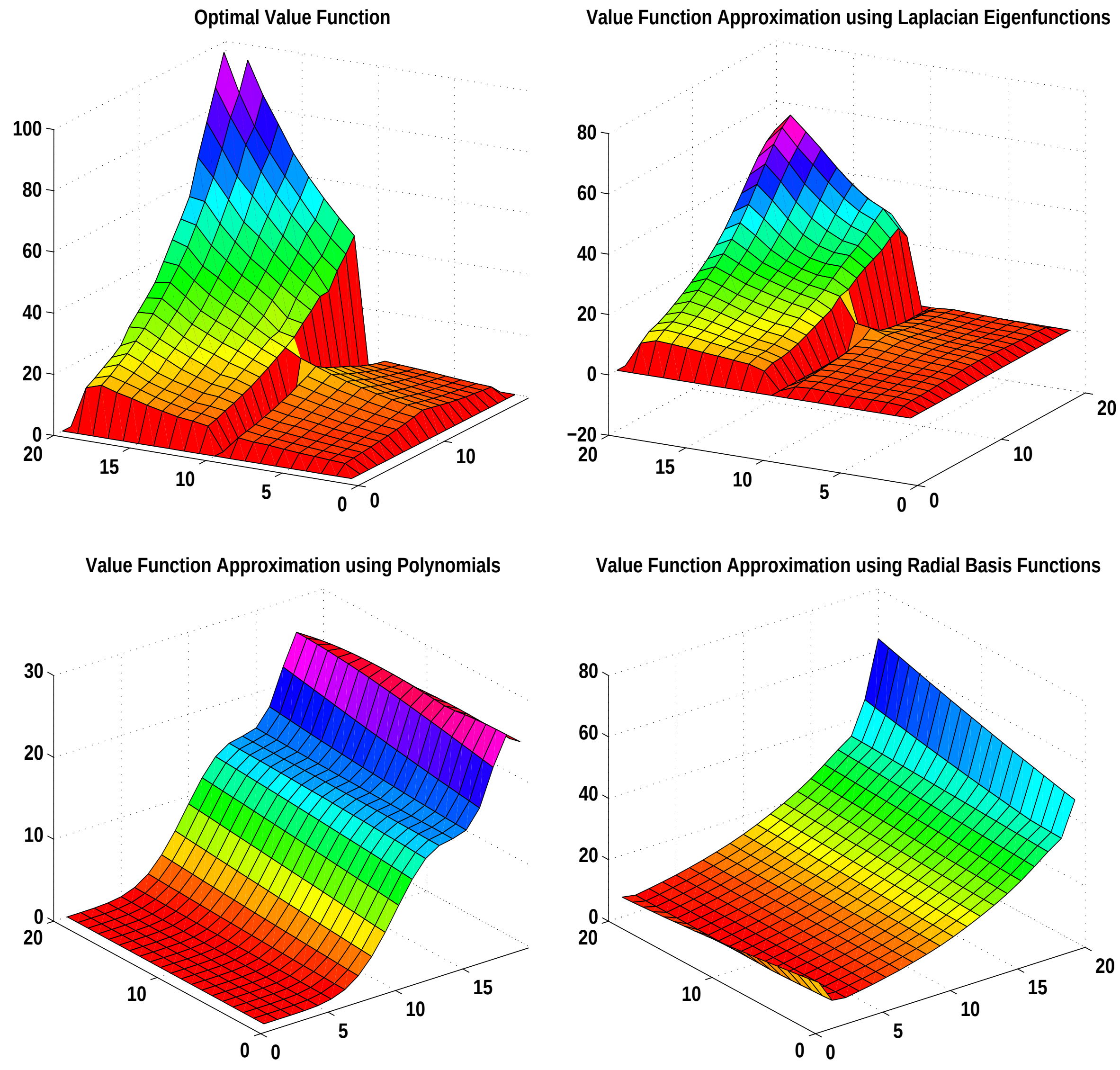


Figure 12: This figure compares the optimal value function for a two-room grid MDP (top left) with approximations produced by RPI (using the Bellman residual approach) with 20 Laplacian eigenfunctions per action (top right); a degree 10 polynomial approximator using LSPI (bottom left); and radial basis function approximator using LSPI with 9 basis functions (bottom right). The value function is nonlinear due to the "bottleneck" region representing the door connecting the two rooms. The Laplacian approximation clearly captures the nonlinearity arising from the bottleneck, whereas the polynomial and the radial basis function approximators smooth the value function across the walls as they are "blind" to the large-scale geometry of the environment. Higher degree polynomial and radial basis function approximators produced significantly poorer approximations.

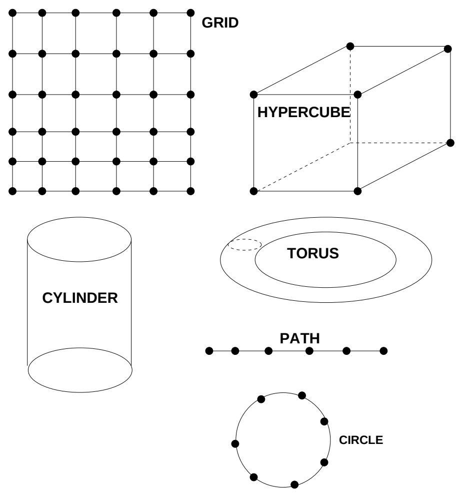


Figure 13: The spectrum and eigenspace of structured state spaces, including grids, hypercubes, cylinders, and tori, can be efficiently computed from "building block" subgraphs, such as paths and circles. Applied to MDPs, this hierarchical framework greatly reduces the computational expense of computing and storing proto-value functions.

framework is now described for efficiently computing and compactly storing proto-value functions that is for all practical purposes *independent* of the size of the state space. Many RL domains lead to *factored* representations where the state space is generated as the cartesian product of the values of state variables (Poupart et al., 2002). Abstractly, consider a hypercube Markov decision process with d dimensions, where each dimension can take on k values. The size of the resulting state space is $O(k^d)$, and the size of each proto-value function is $O(k^d)$. Using the hierarchical framework presented below, the hypercube can be viewed as the *Kronecker sum* of d path or chain graphs, each of whose transition matrix is of size (in the worst case) $O(k^2)$. Now, each factored proto-value function can be stored in space $O(dk^2)$, and the cost of spectral analysis greatly reduces as well. Even greater savings can be accrued since usually only a small number of basis functions are needed relative to the size of a state space. We present detailed experimental results in Section 8 on a large factored multiagent domain of $> 10^6$ states, where proto-value functions are constructed from diagonalizing Laplacian matrices of size only 10×10, a huge computational savings!

Following (Cvetkovic et al., 1980), various compositional schemes can be defined for constructing complex graphs from simpler graphs. We focus on compositions that involve the Kronecker (or the tensor) sum or products of graphs. Let $G_1, \ldots, G_n$ be n undirected graphs whose corresponding vertex and edge sets are specified as $G_i = (V_i, E_i)$. The *Kronecker sum graph* $G = G_1 \oplus \ldots \oplus G_n$ has the vertex set $V = V_1 \times \ldots V_n$, and edge set $E(u, v) = 1$, where $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, if and only if u_k is adjacent to v_k for some $u_k, v_k \in V_k$ and all $u_i = v_i, i \neq k$. For example, the grid graph illustrated in Figure 13 is the *Kronecker sum* of two path graphs; the hypercube is the Kronecker sum of three or more path graphs. In contrast, the *Kronecker product graph* is the graph

$G = G_1 \otimes \ldots G_n$ defined as having a vertex set $V = V_1 \times \ldots V_n$, and whose set of edges is defined as follows: $E(u, v) = 1$ if and only if $E_i(u_i, v_i) = 1$ for all $1 \leq i \leq n$.

The Kronecker sum and product graphs can also be defined using operations on the component adjacency matrices. Given graphs G_1 and G_2, the adjacency matrix of the product graph $G = G_1 \otimes G_2$ is $A = A_1 \otimes A_2$, the *Kronecker product* of the invidual adjacency matrices.[4] If A_1 is an (p, q) matrix and A_2 is a (r, s) matrix, the Kronecker product matrix A is a (pr, qs) matrix, where $A(i, j) = A_1(i, j) * A_2$. In other words, each entry of A_1 is replaced by the product of that entry with the entire A_2 matrix. Similarly, the Kronecker sum of two graphs $G = G_1 \oplus G_2$ can be defined as the graph whose adjacency matrix is the *Kronecker sum* $A = A_1 \otimes I_2 + A_2 \otimes I_1$, where I_1 and I_2 are the identity matrices of size equal to number of rows (or columns) of A_1 and A_2, respectively.

Two more general graph composition schemes, which include the Kronecker sum and product as special cases, are the p-sum and the non-complete extended p-sum (NEPS) (Cvetkovic et al., 1980). The p-sum of a set of graphs $G_1, \ldots, G_n$ is the graph G whose vertex set $V = V_1 \times \ldots V_n$, and whose edge set $E(u, v) = 1$ if and only if exactly p pairs, say $u_i, \ldots, u_p$ and $v_i, \ldots, v_p$, are adjacent vertices in the constituent graphs, namely $(u_i, v_i) \in E_i, \ldots, (u_p, v_p) \in E_p$, and the other $n - p$ pairs are identical vertices. Notice that when $p = 1$, this reduces to the sum of two graphs, and when $p = n$, this reduces to the product of two graphs. Let $\mathcal{B}$ be any set of n-tuples $(\beta_1, \ldots, \beta_n)$ where each $\beta_i \in (0, 1)$, except for the tuple $(0, \ldots, 0)$. The *non-extended complete p-sum* (NEPS) of n graphs $G_1, \ldots, G_n$ is the graph G whose vertex set V is the cartesian product of the vertex sets $V_1, \ldots, V_n$, and whose edge set is defined as follows: $E(u, v) = 1$ if and only if there is an n-tuple $(\beta_1, \ldots, \beta_n) \in \mathcal{B}$ such that $u_i = v_i$ holds exactly when $\beta_i = 0$ and $(u_i, v_i) \in E_i$ exactly when $\beta_i = 1$. Note that for the case of the sum of two graphs, we set $\mathcal{B} = \{(0, 1), (1, 0)\}$ and the product of two graphs is given by $\mathcal{B} = \{(1, 1)\}$.

8.2 Spectra and Eigenspaces of Composite Graphs

Composition	**Operators**
Kronecker Sum	Combinatorial Laplacian
Kronecker Product	Normalized Laplacian, Random Walk
p-sum	Adjacency
NEPS	Adjacency

Table 3: Spectral operators under different composition schemes.

Table 3 summarizes how the different graph operators behave under the compositional schemes introduced above. A brief explanation of some of these results follows.

4. The Kronecker product of two matrices is often also referred to as the *tensor product* in the literature (Chow, 1997).

Theorem 2 *(Cvetkovic et al., 1980) Let $A_1, \ldots, A_n$ be the adjacency matrices of graphs $G_1, \ldots, G_n$. Then, the NEPS of these graphs has the adjacency matrix A specified as*

$$A = \sum_{\beta \in \mathcal{B}} A_1^{\beta_1} \otimes \ldots \otimes A_n^{\beta_n}$$

As an example, if $G = G_1 \oplus G_2$, the sum of two graphs, then its adjacency matrix becomes $A = A_1 \otimes I_2 + A_2 \otimes I_1$ as before (since $\mathcal{B} = \{(0,1),(1,0)\}$) whereas for the product of two graphs $G = G_1 \times G_2$, we get $A = A_1 \otimes A_2$ (since $\mathcal{B} = \{(1,1)\}$). A crucial property we will exploit below is that the eigenvectors of the Kronecker product of two matrices can be expressed as the Kronecker products of the eigenvectors of the component matrices.

Theorem 3 *Let A and B be full rank square matrices of size $r \times r$ and $s \times s$, respectively, whose eigenvectors and eigenvalues can be written as*

$$Au_i = \lambda_i u_i, \ 1 \leq i \leq r \qquad Bv_j = \mu_j v_j, \ 1 \leq j \leq s$$

Then, the eigenvalues and eigenvectors of the Kronecker product $A \otimes B$ and Kronecker sum $A \oplus B$ are given as

$$\begin{aligned} (A \otimes B)(u_i \otimes v_j) &= \lambda_i \mu_j (u_i \otimes v_j) \\ (A \otimes I_s + I_r \otimes B)(u_i \otimes v_j) &= (\lambda_i + \mu_j)(u_i \otimes v_j) \end{aligned}$$

The proof of this theorem relies on the following identity regarding Kronecker products of matrices: $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ for any set of matrices where the products AC and BD are well defined. This theorem forms the basis for several of the results below, starting with the most general one regarding the eigenspace of the NEPS of graphs. Here, we denote the set of eigenvectors of an operator $\mathcal{T}$ by the notation $X(\mathcal{T})$ and its spectrum by $\sigma(\mathcal{T})$. The next result follows immediately from the above theorems.

Theorem 4 *(Cvetkovic et al., 1980) Let $A_1, \ldots, A_n$ be the adjacency matrices of graphs $G_1, \ldots, G_n$, where each individual spectrum and eigenspace is given by $\sigma(A_i) = (\lambda_{i1}, \ldots, \lambda_{in_i})$ and $X(A_i) = (x_{i1}, \ldots, x_{in_i})$. Then, the spectrum and eigenspace of the NEPS of these graphs is given by all possible combinations of the form*

$$\Lambda_{i_1, \ldots, i_n} = \sum_{\beta \in \mathcal{B}} \lambda_{1i_1}^{\beta_1} \ldots \lambda_{ni_n}^{\beta_n} \qquad x \in X = x_{1i_1} \otimes \cdots \otimes x_{ni_n}$$

In effect, the eigenvectors of the adjacency matrix of the NEPS of n graphs is simply the Kronecker product of the eigenvectors of individual adjacency matrices. For the graph Laplacians, this crucial property holds true under more restricted conditions. In particular, the combinatorial graph Laplacian is well-defined for the sum of two graphs:

Theorem 5 *If $L_1 = L(G_1)$ and $L_2 = L(G_2)$ are the combinatorial Laplacians of graphs G_1 and G_2, then the spectral structure of the combinatorial Laplacian $L(G)$ of the Kronecker sum of these graphs $G = G_1 \oplus G_2$ can be computed as*

$$(\sigma(L), X(L)) = \{\lambda_i + \kappa_j, l_i \otimes k_j\}$$

where λ_i is the i^{th} eigenvalue of L_1 with associated eigenvector l_i and κ_j is the j^{th} eigenvalue of L_2 with associated eigenvector k_j.

The proof is omitted, but fairly straightforward by exploiting the property that the Laplace operator acts on a function by summing the difference of its value at a vertex with those at adjacent vertices. In contrast, the normalized Laplacian is not well-defined under sum, but has a well-defined semantics for the product of two graphs:

Theorem 6 *If $\mathcal{L}_1 = \mathcal{L}(G_1)$ and $\mathcal{L}_2 = \mathcal{L}(G_2)$ be the normalized Laplacians of graphs G_1, G_2, then the spectral structure of the normalized Laplacian $L(G)$ of the product of these graphs $G = G_1 \otimes G_2$ can be computed as*

$$(\sigma(\mathcal{L}), X(\mathcal{L})) = \{\lambda_i \times \kappa_j, l_i \otimes k_j\}$$

where λ_i is the i^{th} eigenvalue of $\mathcal{L}_1$ with associated eigenvector l_i and κ_j is the j^{th} eigenvalue of $\mathcal{L}_2$ with associated eigenvector k_j.

Not surprisingly, the random walk operator is well-behaved for graph products as well. Both these results exploit the property that the adjacency matrix of the product graph is the Kronecker product of the adjacency matrices of the individual graphs. One can easily show that the random walk stochasticization term $S(A) = D^{-1}A$ distributes over the Kronecker product of matrices (Chow, 1997):

$$S(A_1 \otimes \cdots \otimes A_n) = S(A_1) \otimes \cdots \otimes S(A_n)$$

and the result follows immediately by applying the above theorem regarding the spectrum of the Kronecker product of a set of matrices. For both Laplacians, the constant vector $\mathbf{1}$ is an eigenvector with associated eigenvalue $\lambda_0 = 0$. Since the eigenvalues of the Kronecker sum graph are the sums of the eigenvalues of the individual graphs, 0 will be an eigenvalue of the Laplacian of the sum and product graphs as well. Furthermore, for each eigenvector v_i, the Kronecker product $v_i \otimes \mathbf{1}$ will also be an eigenvector of the sum and product graphs. One consequence of these properties is that geometry is well preserved, so for example the Laplacians produce well-defined embeddings of structured spaces. Figure 14 shows the embedding of a cylinder (Kronecker sum of a closed and open chain) under the combinatorial Laplacian.

8.3 Factored Representation Policy Iteration for Structured Domains

We derive the update rule for a factored form of RPI (and LSPI) for structured domains when the basis functions can be represented as Kronecker products of elementary basis functions on simpler state spaces. Basis functions are *column* eigenvectors of the diagonalized representation of a spectral operator, whereas embeddings $\phi(s)$ are *row* vectors representing the first k basis functions evaluated on state s. By exploiting the property that $(A \otimes B)^T = A^T \otimes B^T$, it follows that embeddings for structured domains can be computed as the Kronecker products of embeddings for the constituent state components. As a concrete example, a grid world domain of size $m \times n$ can be represented as a graph $G = G_m \oplus G_n$ where G_m and G_n are *path graphs* of size m and n, respectively. The basis functions for the entire grid world can be written as the Kronecker product $\phi(s) = \phi_m(s^r) \otimes \phi_n(s^c)$, where $\phi_m(s^r)$ is the basis (eigen)vector derived from a path graph of size m (in particular, the row s^r corresponding to state s in the grid world), and $\phi_n(s^c)$ is the basis (eigen)vector derived

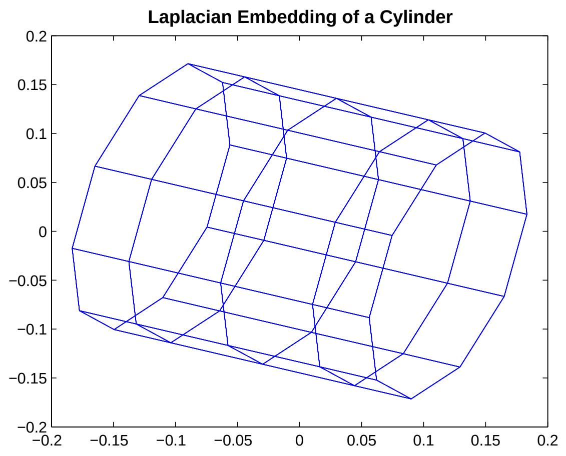


Figure 14: This figure shows the embedding of a cylinder using two low-order eigenvectors (3^{rd} and 4^{th}) of the combinatorial Laplacian. The cylinder is modeled as the Kronecker sum of a closed and open chain graph.

from a path graph of size n (in particular, the column s^c corresponding to state s in the grid world).

Extending this idea to state action pairs, the basis function $\phi(s,a)$ can written as $e_I(a) \otimes \phi(s)$, where $e_I(a)$ is the unit vector corresponding to the index of action a (e.g., action a_1 corresponds to $e_1 = [1,0,\ldots]^T$). Actually, the full Kronecker product is not necessary if only a relatively small number of basis functions are needed. For example, if 50 basis functions are to be used in a $10 \times 10 \times 10$ hypercube, the full state embedding is a vector of size 1000, but only the first 50 terms need to be computed. Such savings imply proto-value functions can be efficiently computed even in very large structured domains. For a factored state space $s = (s^1, \ldots, s^m)$, we use the notation s^i to denote the value of the i^{th} component. We can restate the update rules for factored RPI and LSPI as follows:

$$\begin{aligned}
\tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t,a_t)\left(\phi(s_t,a_t) - \gamma\phi(s'_t,\pi(s'_t))\right)^T \\
&= \tilde{A}^t + e_{I(a_t)} \otimes \prod_{\otimes} \phi_i(s^i_t) \\
&\times \left(e_{I(a_t)} \prod_{\otimes} \phi_i(s^i_t) - \gamma e_{I(\pi(s'_t))} \otimes \prod_{\otimes} \phi_i({s'}^i_t) \right)^T \\
&= \tilde{A}^t + (e_{I(a_t)}(e_{I(a_t)})^T) \otimes \prod_{\otimes}(\phi_i(s^i_t)\phi_i(s^i_t)^T) - \gamma(e_{I(s_t)}(e_{I(s'_t)})^T) \otimes \prod_{\otimes}(\phi_i(s^i_t)\phi_i({s'}^i_t)^T)
\end{aligned}$$

The corresponding update equation for the reward component is:

$$\begin{aligned}
\tilde{b}^{t+1} &= \tilde{b}^t + \phi(s_t,a_t)r_t \\
&= \tilde{b}^t + r_t e_{I(a_t)} \otimes \prod_{\otimes} \phi_i(s^i_t)
\end{aligned}$$

8.4 Experimental Results

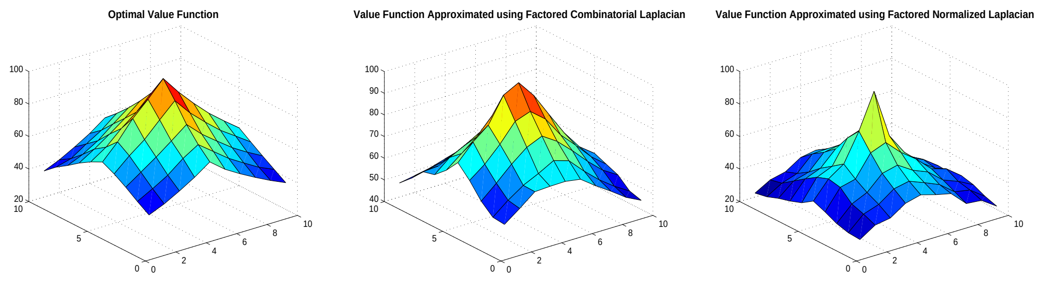


Figure 15: The figure on the left is the exact value function on a 10×10 grid world with a reward of +100 at the center. In the middle is the factored (combinatorial) Laplacian approximation using basis functions constructed by taking tensor products of basis functions for chain graphs (of length corresponding to row and column sizes). On the right is the similar approximation using the normalized Laplacian basis functions, which is significantly poorer since the eigenvectors of the normalized Laplacian of a grid cannot be decomposed as the tensor product of the eigenvectors of the normalized Laplacian on chain graphs.

Figure 15 shows the results of using the factored RPI algorithm on a 10×10 grid world domain. There are four (compass direction) actions, each of which succeeds with probability 0.9. Any "illegal" action (going "north" from the first row) leaves the agent in the same state. The only reward of +100 is received for reaching the center of the grid. The discount factor was set at $\gamma = 0.9$. If a "flat" approach was used, each basis function is a vector of size 100 and requires diagonalizing a Laplacian matrix of size 100×100. The factored proto-value functions are computed as the Kronecker product of the proto-value functions on a 10 node chain graph, which requires both significantly smaller space of $\mathcal{O}(10)$ and much less computational effort (diagonalizing a Laplacian of size 10×10). These computational savings obviously magnify in larger grid world domains (e.g., in a grid world with 10^6 states, "flat" proto-value functions require $\mathcal{O}(10^6)$ space and time $\mathcal{O}((10^6)^3)$ to be computed, whereas the factored basis functions only require space $\mathcal{O}(10^3)$ to store with much less computational cost to find).

We now present a detailed study using a much larger factored multiagent domain called the "blockers" task (Sallans and Hinton, 2004). This task, shown in Figure 16, is a cooperative multiagent problem where a group of agents try to reach the top row of a grid, but are prevented in doing so by "blocker" agents who move horizontally on the top row. If any agent reaches the top row, the entire team is rewarded by +1; otherwise, each agent receives a negative reward of −1 on each step. The agents always start randomly placed on the bottom row of the grid, and the blockers are randomly placed on the top row. The blockers remain restricted to the top row, executing a fixed strategy. The overall state space is the cartesian product of the location of each agent. Our experiments on the blocker domain include more difficult versions of the task not studied in (Sallans and Hinton, 2004) specifically designed to test the scalability of the pure tensor product bases to "irregular" grids whose topology deviates from a pure hypercube or toroid. In the first variant, shown on the left in Figure 16, horizontal interior walls extend out from the left and right side walls between the second and third row. In the second variant, an additional interior wall is added in the middle as shown on the right. Both of these variants are "perturbations" of the pure product of grids or cylinders topology generated by the original "obstacle-free" blocker domain designed to test the robustness of the tensor product approach.

The basis functions for the overall blockers state space were computed as tensor products of the basis functions over each agent's state space. Basis functions for each agent were computed in two ways. In the simplest "pure" approach, each agent's state space was modeled as a grid (as in Figure 15) or a cylinder (for the "wrap-around" case); the latter case is modeled as a Kronecker sum graph of a "column" open chain and a "row" cycle graph. In effect, the overall basis functions were then computed as a two-level nested tensor product of the basis functions for the "row" chain and the "column" cylinder. The presence of interior walls obviously violates the pure product of cylinders or grids topology. We compared the performance of the "pure" two-level nested tensor approach with a one-level tensor product approach, where an undirected graph representing the irregular grid was first learned from a random walk, and then basis functions for the overall state space were constructed as tensor products of Laplacian basis functions for each learned irregular state grid.

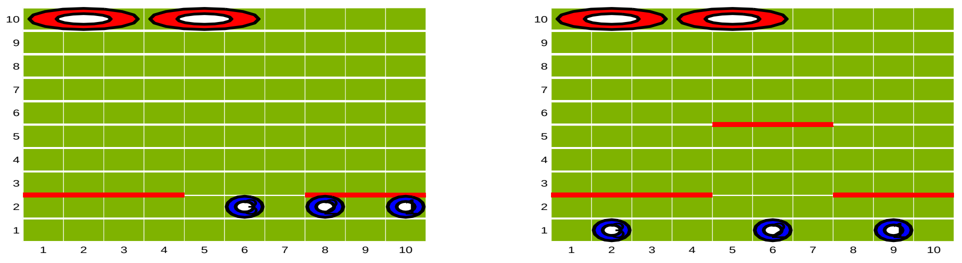


Figure 16: Two versions of the blocker domain are shown, each generating a state space of $> 10^6$ states. Interior walls shown create an "irregular" factored MDP whose overall topology can be viewed as a "perturbed" variant of a pure product of grids or cylinders (for the "wrap-around" case).

Figure 17 compares the performance of the "pure" two-level nested tensor product approach and the single-level tensor product Laplacian bases with a set of radial basis functions (RBFs). The width of each RBF was set at $\frac{2|S_a|}{k}$ where $|S_a|$ is the size of each individual agent's grid, and k is the number of RBFs used. The RBF centers were uniformly spaced. The results shown are are averages over 10 learning runs. On each run, the learned policy is measured every 25 training episodes. Each episode begins with a random walk of a maximum of 70 steps (terminating earlier if the top row was reached). After every 25 such episodes, RPI is run on all the samples collected thus far. The learned policy is then tested over 500 test episodes. The graphs plot the probability of termination within a maximum of 30 steps as well as the average number of steps to reach the goal. The experiments were conducted on both "normal" grids (not shown) and "wrap around" cylindrical grids. The results show that RBFs converge the fastest, but learn the worst policy. The single-level nested tensor product Laplacian bases converge slower than RBFs, but learn a substantially better policy. Finally, the simplest "pure" tensor Laplacian bases converges the slowest, but surprisingly learns the best policy.

Figure 18 shows results for the second 10×10 blockers task with both side and interior middle walls, comparing 100 single-level tensor and nested two-level tensor factored Laplacian bases with a similar number of RBFs. The results show a significant improvement in performance of of both factored Laplacian bases over RBFs. Furthermore, although the "pure" nested tensor bases converge more slowly than the one-level tensor bases, the ultimate performance is almost as good, indicating little loss due to the perturbations caused by the interior and middle walls. In terms of space, the two-level nested tensor product approach greatly reduces the computational complexity of finding and storing the Laplacian bases of five orders of magnitude, from $O(10^6)$ space for the non-factored approach and $O(10^2)$ for the one-level factored approach to $O(10)$ for the two-level nested tensor product approach! The two-level tensor bases scale polynomially with the *row* and *column* size of a *single* grid (or cylinder). The one-level generates larger basis functions, scaling polynomially in the size of each grid or cylinder. Both of these tensor approaches scale more efficiently than the non-tensor "flat" approach.

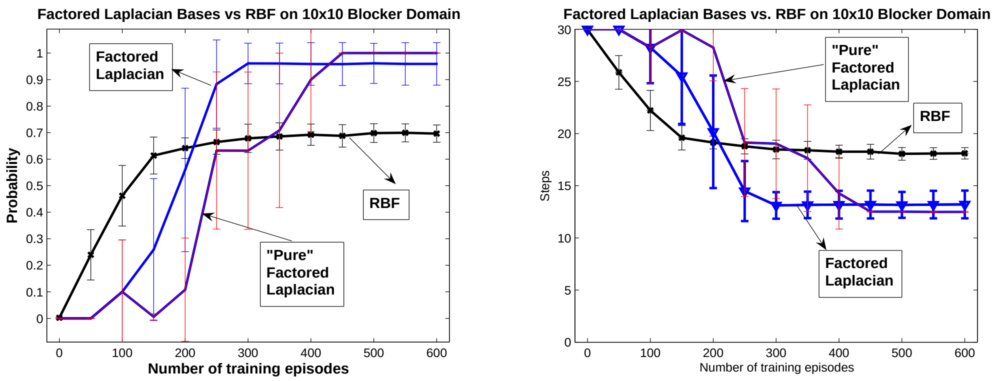


Figure 17: Comparison of two types of factored Laplacian basis functions with hand coded radial basis functions (RBF) on a 10×10 "wrap-around" grid with 3 agents and 2 blockers of $> 10^6$ states. All three approaches compared using 100 basis functions.

Why does the pure tensor approach perform so well even in irregular structured MDPS, as shown above? A detailed analysis requires the use of matrix perturbation analysis (Stewart and Sun, 1990), which is beyond the scope of this paper. However, Figure 19 provides some insight. In the blockers task, the state space topologically can be modeled as the product of cylinders (when the grid has "wrap-around-dynamics"). The Laplacian spectrum of the product of cylinders has many repeated eigenvalues, which are clustered together. A standard result in matrix perturbation analysis shows that under these conditions the eigenspace defined by the eigenvectors associated with eigenvalues with high geometric multiplicity are fairly robust with respect to perturbations.

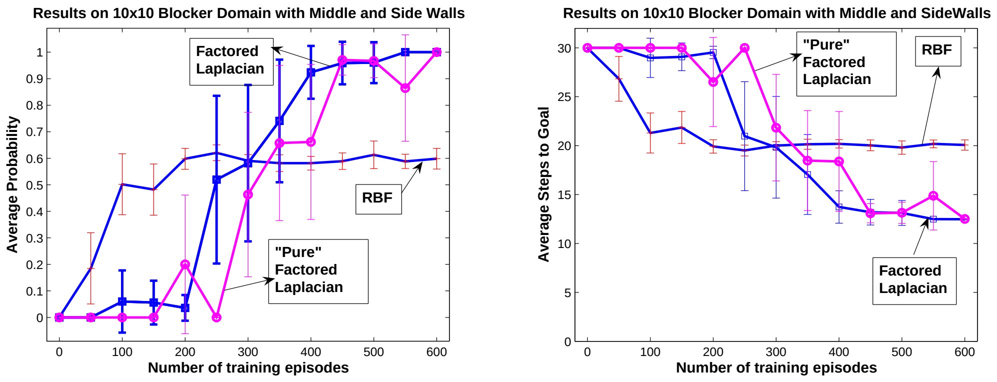


Figure 18: Performance on the second more irregular blockers domain comparing 100 factored one-level and two-level tensor product Laplacian basis functions with 100 RBFs.

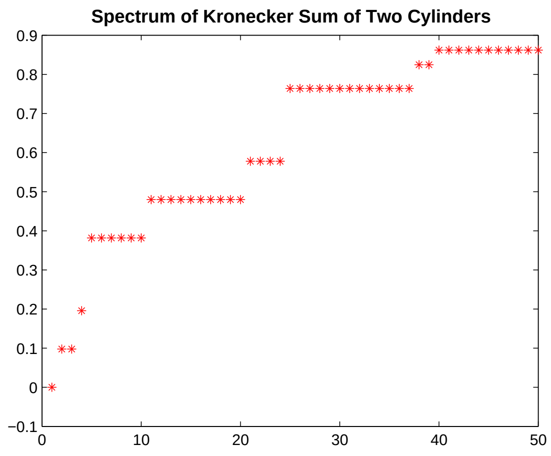


Figure 19: This figure shows the first 50 eigenvalues in the spectrum of the Kronecker sum of two cylinders, revealing that many of the eigenvalues are of high algebraic multiplicity.

9. Scaling Proto-Value Functions to Continuous Domains

Thus far, the construction of proto-value functions was restricted to discrete MDPs. We now show how proto-value functions can be constructed for continuous MDPs, which present significant challenges not encountered in discrete state spaces. The eigenfunctions of the Laplacian can only be computed and stored on sampled real-valued states, and hence must be interpolated to novel states. We apply the Nyström interpolation method. While this approach has been studied previously in kernel methods (Williams and Seeger, 2000) and spectral clustering (Belongie et al., 2002), our work represents the first detailed study of the Nyström method for learning control, as well as a detailed comparison of graph normalization methods (Mahadevan et al., 2006).

9.1 Nyström Extension

To learn policies on continuous MDPs, it is necessary to be able to extend eigenfunctions computed on a set of points $\in \mathbb{R}^d$ to new unexplored points. We describe here the Nyström method, which can be combined with iterative updates and randomized algorithms for low-rank approximations. The Nyström method interpolates the value of eigenvectors computed on sample states to novel states, and is an application of a classical method used in the numerical solution of integral equations (Baker, 1977). It can be viewed as a technique for approximating a semi-positive definite matrix from a low-rank approximation. In this context it can be related to randomized algorithms for low-rank approximation of large matrices (Frieze et al., 1998). Let us review the Nyström method in its basic form. Suppose we have a positive semi-definite operator K, with rows and columns indexed by some measure space $(\mathbb{X}, \mu)$. K acts on a vector space of functions on X by the formula

$$Kf(x) = \int_{\mathbb{X}} K(x, y) f(y) d\mu(y) \,,$$

for f in some function space on $\mathbb{X}$. Examples include:

(i) $\mathbb{X} = \mathbb{R}$, μ is the Lebesgue measure, and $K_\sigma(x, y) = e^{-\frac{|x-y|^2}{\sigma}}$, and K acts on square integral functions f on $\mathbb{R}$ by $K_\sigma f(x) = \int_{-\infty}^{+\infty} e^{-\frac{|x-y|^2}{\sigma}} f(y) dy = K_\sigma * f$.

(ii) $\mathbb{X}$ is a compact Riemannian manifold $(\mathcal{M}, \rho)$ equipped with the measure corresponding to the Riemannian volume, Δ is the Laplace-Beltrami operator on $\mathcal{M}$, with Dirichlet or Neumann boundary conditions if $\mathcal{M}$ has a boundary, and $K = (I - \Delta)^{-1}$ is the Green's function or potential operator associated with Δ.

Since K is positive semi-definite, by the spectral theorem it has a square root F, i.e. $K = F^T F$. Sometimes this property is expressed by saying that K is a Gram matrix, since we can interpret $K(x, y)$ as the inner product between the x-th and y-th columns of F. In applications one approximates operators on uncountable spaces (such as $\mathbb{R}$ or a manifold $\mathcal{M}$ as in the examples above) by a finite discretization $x_1, \ldots, x_n$, in which case $\mathbb{X} = \{0, \ldots, n\}$, the measure μ is an appropriate set of weights on the n points, and K is a $n \times n$ matrix acting on n-dimensional vectors. To simplify the notation we use this discrete setting in what follows.

The Nyström approximation starts with a choice of a partition of the columns of F into two subsets F_1 and F_2. Let k be the cardinality of F_1, so that F_1 can be represented as $n \times k$ matrix and F_2 as a $n \times (n-k)$ matrix. One can then write

$$K = \begin{pmatrix} F_1^T F_1 & F_1^T F_2 \\ F_2^T F_1 & F_2^T F_2 \end{pmatrix}$$

The Nyström method consists of the approximation

$$F_2^T F_2 \sim (F_1^T F_2)^T (F_1^T F_1)^{-1} (F_1^T F_2) \,. \tag{12}$$

The quantity on the righthand side requires only the knowledge of $(F_1^T F_2)$ and $F_1^T F_1$, i.e. the first k rows (or columns) of K. Moreover if the matrix K has rank k and F_1 spans the range of K, then the Nyström approximation is in fact exactly equal to $F_2^T F_2$.

This technique applies to the discretization of integral equations (Baker, 1977), where the k points F_1 can be chosen according to careful mathematical and numerical analysis of the problem, and has been applied to speeding up the computations in learning and clustering algorithms (Platt, 2004; Williams and Seeger, 2000; Belongie et al., 2002). The natural question that arises is of course how to choose F_1 in these situations. Various heuristics exist, and mixed results have been obtained (Platt, 2004). The most desirable choice of F_1, when the error of approximation is measured by $||F_2^T F_2 - (F_1^T F_2)^T (F_1^T F_1)^{-1} (F_1^T F_2)||_2$ (or, equivalently, the Fröbenius norm) would be to pick F_1 such that its span is as close as possible to the span of the top k singular vectors of K. Several numerical algorithms exist, which in general require $\mathcal{O}(kN^2)$ computations. One can use randomized algorithms, which pick rows (or columns) of K accordingly to some probability distribution (e.g. dependent on the norm of the row or column). There are guarantees that these algorithms will select with high probability a set of rows whose span is close to that of the top singular vectors: see for example (Frieze et al., 1998; Drineas et al., 2004; Drineas and Mahoney, 2005).

The Nyström method is applied to the approximation of the eigenfunctions of the graph Laplacian $\mathcal{L}\phi_i = \lambda_i \phi_i$ by letting F_1 be the matrix with k eigenfunctions as columns: equation (12) yields

$$\phi_i(x) = \frac{1}{1 - \lambda_i} \sum_{y \sim x} \frac{w(x,y)}{\sqrt{d(x)d(y)}} \phi_i(y) \,, \tag{13}$$

where $d(z) = \sum_{y \sim z} w(z,y)$, and x is a new vertex in the graph. The Nyström method can be refined with fast iterative updates as follows: first compute an extension of the eigenvectors to new points (states), to obtain approximated eigenvectors of the extended graph $\{\tilde{\phi}_i\}$. Input these eigenvectors into an iterative eigensolver as initial approximate eigenvectors: after very few iterations the eigensolver will refine these initial approximate eigenvectors into more precise eigenvectors on the larger graph. The extra cost of this computation is $\mathcal{O}(IN)$ if I iterations are necessary, and if the adjacency matrix of the extended graph is sparse (only N non-zero entries).

9.2 Representation Policy Iteration for Continuous Domains

Figure 21 presents the modified RPI algorithm for continuous Markov decision processes. The core of the algorithm remains the same as before, but there are important differences from the discrete case. First, the proto-value functions are computed on a subsampled set of states, for two reasons: the number of samples needed to compute the proto-value functions is much less than that needed to learn a good policy using RPI, as the experiments in Section 9.5 reveal. In Figure 21, $\mathcal{D}_{\mathcal{Z}}$ denotes the subsampled set of states. For example, in the inverted pendulum, typically fewer than 500 state samples are sufficient to compute a good set of proto-value functions. On the other hand, to learn a good policy requires many more samples. The choice of the subsampling method can make a significant difference, as explained below. The second major difference is the use of the Nyström method to extend proto-value functions from the samples stored in $\mathcal{D}_{\mathcal{Z}}$ to all the states visited during the initial random walk (denoted $\mathcal{D}$ in Figure 21), as well as new states encountered during the testing of a learned policy.

9.3 Sampling from Point Sets $\in \mathbb{R}^d$

One challenge in continuous MDPs is how to choose a subset of samples from which a graph can be built and proto-value functions computed. The set of samples collected during the course of exploration and random walk can be very large, and a much smaller set of samples is usually sufficient to learn proto-value functions. Many ways of constructing a subsample from the overall sample can be devised. The simplest method is of course to randomly subsample from the complete set, but this might not be the most efficient way of using the samples. Figure 20 illustrates two methods for subsampling in the mountain car domain (described below in more detail), including random subsampling and trajectory-based subsampling. The trajectory-based method also tries to retain "important" samples, such as goal states or states with high reward. Note that the random subsampling method clearly loses important information about the trajectory, which is nicely retained by the trajectory method.

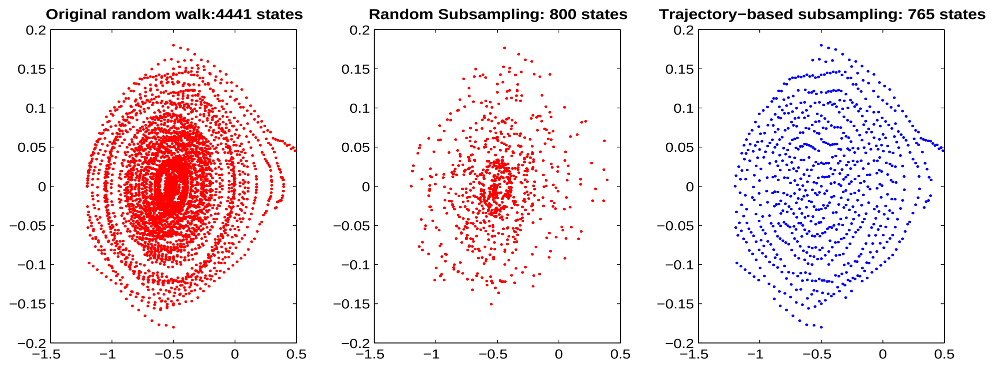


Figure 20: The problem of subsampling is illustrated in the mountain car domain. On the left is shown the original states visited during a random walk. In the middle is the subsampled data using a random subsampling algorithm. On the right is a trajectory based subsampling method.

9.4 Graph Construction from Point Sets $\in \mathbb{R}^d$

Given a data set $\{x_i\}$ in $\mathbb{R}^d$, we can associate different weighted graphs to this point set. There are different choices of edges and for any such choice there is a choice of weights on the edges. For example, edges can be inserted between a pair of points x_i and x_j if:

(E1) $||x_i - x_j||_{\mathbb{R}^d} \leq \delta$, where $\delta > 0$ is a parameter;

(E2) x_j is among the k nearest neighbors of x_i, where $k > 0$ is a parameter.

Weights can be assigned to the edges in any of the following ways:

```
RPI Algorithm $(T, N, Z, \epsilon, k, P, \mathcal{O})$:

// $T$: Number of initial random walk trials
// $N$: Maximum length of each trial
// $Z$: Size of the subsampled data to build the graph from
// $\epsilon$ : Convergence condition for policy iteration
// $k$: Number of nearest neighbors
// $P$: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used

  1. Representation Learning Phase: Perform a random walk of $T$ trials, each of maximum
     $N$ steps, and store the states visited in the dataset $\mathcal{D}$. Construct a smaller dataset $\mathcal{D}_{\mathcal{Z}}$ of
     size $Z$ from the collected data $\mathcal{D}$ by some sampling method, such as random subsampling or
     trajectory-based subsampling (see Section 9.3).
  2. Build an undirected weighted graph $G$ from $\mathcal{D}_{\mathcal{Z}}$, in one of the ways described in the section
     on the construction of graphs from point sets. Construct one of the operators $\mathcal{O}$ on graph $G$
     as discussed in Section 9.4.
  3. Compute the $k$ "smoothest" eigenvectors of $\mathcal{O}$ on the sub-sampled graph $\mathcal{D}_{\mathcal{Z}}$, and collect them
     as columns of the basis function matrix $\Phi$, a $|\mathcal{D}_{\mathcal{Z}}| \times k$ matrix. The embedding of a state action
     pair $\phi(s, a)$ where $s \in \mathcal{D}_{\mathcal{Z}}$ is given as $e_a \otimes \phi(s)$, where $e_a$ is the unit vector corresponding to
     action $a$, $\phi(s)$ is the $s^{th}$ row of $\Phi$, and $\otimes$ is the tensor product.
  4. Control Learning Phase: Initialize $w^0 \in \mathcal{R}^k$ to a random vector. Repeat the following
     steps
     (a) Set $i \leftarrow i+1$. For each transition $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}$, compute low rank approximations
         of matrix $A$ and $b$ as follows:

             $\tilde{A}^{t+1} = \tilde{A}^t + \phi(s_t, a_t)\left(\phi(s_t, a_t) - \gamma\phi(s'_t, a'_t)\right)^T$
             $\tilde{b}^{t+1} = \tilde{b}^t + \phi(s_t, a_t) r_t$

         where $\phi(s_t, a_t)$ is approximated using the Nyström extension whenever $s_t \notin \mathcal{D}_{\mathcal{Z}}$.
     (b) Solve the system $\tilde{A}w^i = \tilde{b}$
  5. until $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
  6. Return $\hat{Q}^\pi = \sum_i w^i \Phi$ as the approximation to the optimal value function.
end
```

Figure 21: Pseudo-code of the representation policy iteration algorithm for continuous MDPs.

(W1) all edges have the same weight (say, 1);

(W2) $W(i,j) = \alpha(i)e^{-\frac{||x_i - x_j||^2_{\mathbb{R}^d}}{\sigma}}$, where $\sigma > 0$ is a parameter, and α a weight function to be specified;

(W3) $W(i,j) = \alpha(i)e^{-\frac{||x_i - x_j||^2_{\mathbb{R}^d}}{||x_i - x_{k(i)}|| \, ||x_j - x_{k(j)}||}}$ where $k > 0$ is a parameter and $x_{k(i)}$ is the k-th nearest neighbor of x_i (this is called self-tuning Laplacian (L. Zelnik-Manor, 2004)). Again α a weight function to be specified.

Observe that in case (E2) the graph is in general not undirected, since x_j can be among the K nearest neighbors of x_i but x_i may not be among the K nearest neighbors of x_j. Since here we consider undirected graphs, in such cases we replace the weight matrix W constructed so far by the symmetric $W + W^T$, WW^T or W^TW. If the points $\{x_i\}$ are drawn uniformly from a Riemannian manifold, then it is shown in (Belkin and Niyogi, 2004) that (E1)+(W2), with $\alpha = 1$, approximates the continuous Laplace-Beltrami operator on the underlying manifold. If $\{x_i\}$ is not drawn uniformly from the manifold, as it typically happens here when the space is explored by an agent, it is shown in (Lafon, 2004) that a pre-processing normalization step can (must) be performed that yields the weight function α, so that (E1)+(W2) yields an approximation to the Laplace-Beltrami operator. In the experiments below, we use the following terminology:

1. 'Ave' means that the eigenvectors of $D^{-1}(D - W)$ are computed (the discrete Laplacian). This applies to any combination of (E1,2)-(W1-3)
2. 'GraphMarkov' means that the eigenvectors of the normalized graph Laplacian are computed, with any combination of (E1,2)-(W1-3)
3. 'Beltrami' applies to any combination of (E1,2)-(W1-3), however the only theorem known about approximation to the continuous Laplace-Beltrami is for the combination (E1) (with large δ) together with (W2).

9.5 Experimental Results on Continuous Markov Decision Processes

This section presents detailed experimental results of the RPI algorithm illustrated in Figure 21 using benchmark continuous control tasks, including the inverted pendulum and the mountain car. The experiments test the variance in performance with respect to various parameters, such as the local distance metric used, and also compare the performance of RPI with using handcoded function approximators, such as radial basis functions.

The Inverted Pendulum: The inverted pendulum problem requires balancing a pendulum of unknown mass and length by applying force to the cart to which the pendulum is attached. We used the implementation described in (Lagoudakis and Parr, 2003). The state space is defined by two variables: θ, the vertical angle of the pendulum, and $\dot{\theta}$, the angular velocity of the pendulum. The three actions are applying a force of -50, 0, or 50 Newtons. Uniform noise from -10 and 10 is added to the chosen action. State transitions are defined by the nonlinear dynamics of the system, and depend upon the current state and the noisy control signal, u.

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml\dot{\theta}^2 \sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)} \tag{14}$$

where g is gravity, 9.8 m/s^2, m is the mass of the pendulum, 2.0 kg, M is the mass of the cart, 8.0 kg, l is the length of the pendulum, .5 m, and $\alpha = 1/(m+M)$. The simulation time step is set to 0.1 seconds. The agent is given a reward of 0 as long as the absolute value of the angle of the pendulum does not exceed $\pi/2$. If the angle is greater than this value the episode ends with a reward of -1. The discount factor was set to 0.95. The maximum number of episodes the pendulum was allowed to balance was fixed at 3000 steps. Each learned policy was evaluated 10 times.

Figure 22 illustrates a sample run of the RPI algorithm for the inverted pendulum task. The initial random walk was carried out for a period of 300 episodes, each of maximum length 30 steps. The actual number of transitions collected was 2744 steps, indicating that the average length of an episode was around 9 steps until the pole was dropped. The graph was constructed by randomly subsampling 500 states from this larger data set, as shown in the figure. The discount factor was set at 0.9. The graph operator used in this run was the "beltrami" operator described above. The number of nearest neighbors was $k = 50$. 50 proto-value functions were used to approximate the action value function for each action. Figure 23 illustrates four sample proto-value functions that were learned for the inverted pendulum task. Finally, Figure 24 shows the quality of the learned policy, measured by the number of steps the pole was successfully balanced (up to a maximum of 3000 steps), averaged over 20 learning runs.

Mountain Car The goal of the *mountain car* task is to get a simulated car to the top of a hill as quickly as possible (Sutton and Barto, 1998). The car does not have enough power to get there immediately, and so must oscillate on the hill to build up the necessary momentum. This is a minimum time problem, and thus the reward is -1 per step. The state space includes the position and velocity of the car. There are three actions: full throttle forward (+1), full throttle reverse (-1), and zero throttle (0). Its position, x_t and velocity $\dot{x}_t$, are updated by

$$x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}] \tag{15}$$

$$\dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t + -0.0025cos(3x_t)], \tag{16}$$

where the bound operation enforces $-1.2 \leq x_{t+1} \leq 0.6$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. The episode ends when the car successfully reaches the top of the mountain, defined as position $x_t >= 0.5$. In our experiments we allow a maximum of 500 steps, after which the task is terminated without success. The discount factor was set to 0.99.

Figure 25 illustrates some Laplacian basis functions learned for the mountain car domain. Figure 24 illustrates a sample learning run, averaged over 20 trials. Each trial was carried out for a maximum of 500 episodes, each episode was for a maximum of 90 steps unless the goal was reached before then.

Figure 26 compares the approximated value functions using proto-value functions with 108 radial basis functions (placed in an uniform 6×6 array for each of the 3 actions with widths automatically computed from the data), showing that proto-value functions learn

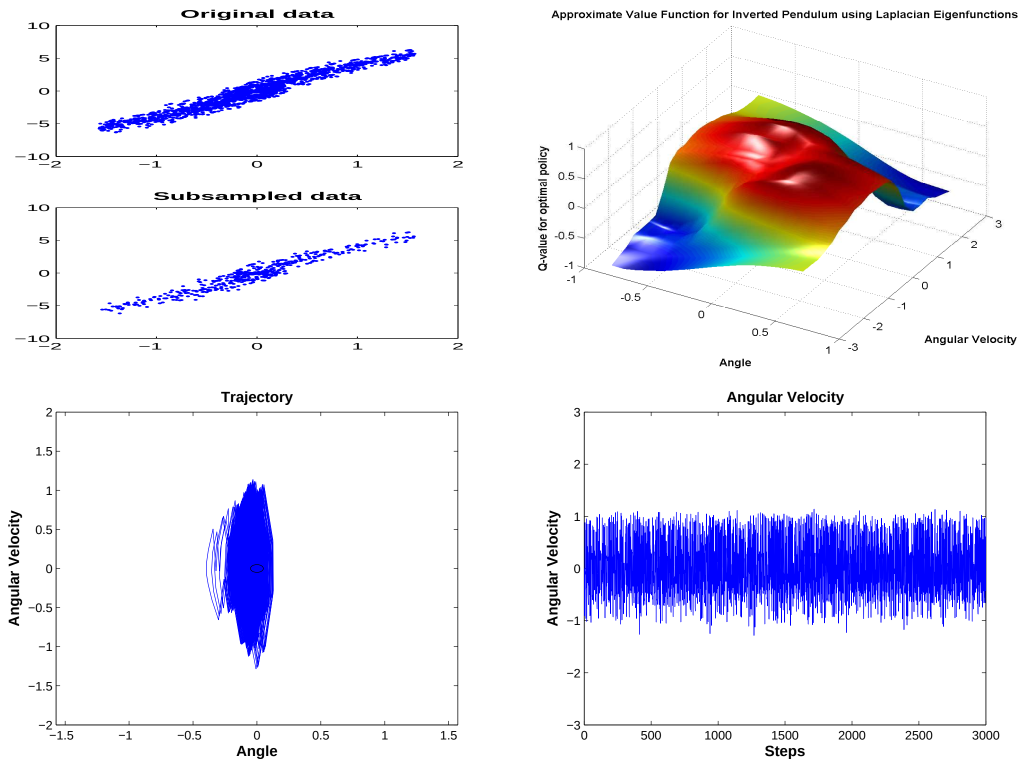


Figure 22: Sample run of RPI on the inverted pendulum, showing (in row major order) the subsampled data from the complete random walk, the action value function for the action of applying a thrust of -50 to the left, a trajectory of 3000 showing the learned policy successfully balancing the pole, and the corresponding angular velocity.

an approximation that is much richer. To do a more thorough comparison, the next section presents a detailed sensitivity analysis for both the inverted pendulum and mountain car domains. Here, the discount factor was set at 0.99. The graph operator used in this run was the "ave" operator described above. The number of nearest neighbors was $k = 20$. 30 proto-value functions were used to approximate the action value function for each action.

9.6 Perturbation Analysis

Given that proto-value functions are a non-parametric framework for constructing basis functions, it is natural to expect that they would be less *biased* than parametric methods, but suffer from a larger degree of *variance*. The goal of this section is to explore this hypothesis by varying the key parameters in the RPI algorithm. Table 4 summarizes the range of parameters over which the RPI algorithm was tested in the two continuous domains. Values in boldface are the defaults for experiments in which those parameters were not being varied. The results for the following experiments were averaged over 20 runs. Table 5 specifies the settings of the parametric radial basis function architecture that was used in

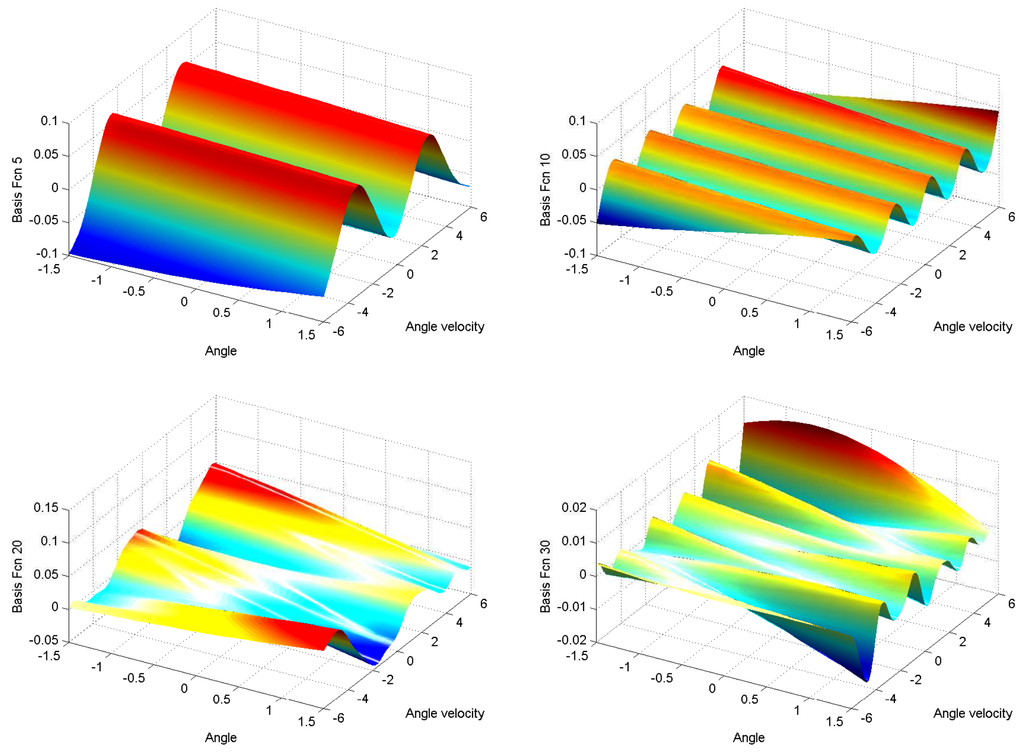


Figure 23: Proto-value functions learned by RPI on the inverted pendulum, showing (in row major order), the basis functions numbered 5, 10, 20, and 30 associated with applying a thrust of −50.

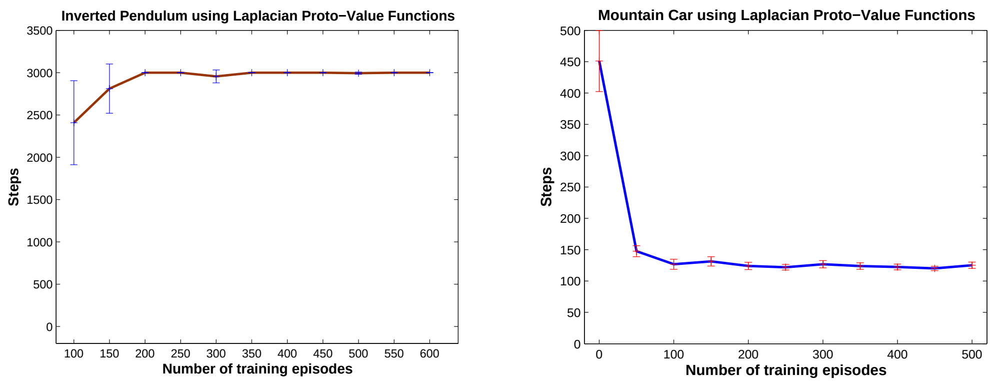


Figure 24: Results of a learning run of RPI on the inverted pendulum task (left) and the mountain car task (right) using Laplacian proto-value functions.

the experiments. For the purpose of these experiments, a random subsampling method was used to build the graph from which proto-value functions were extracted.

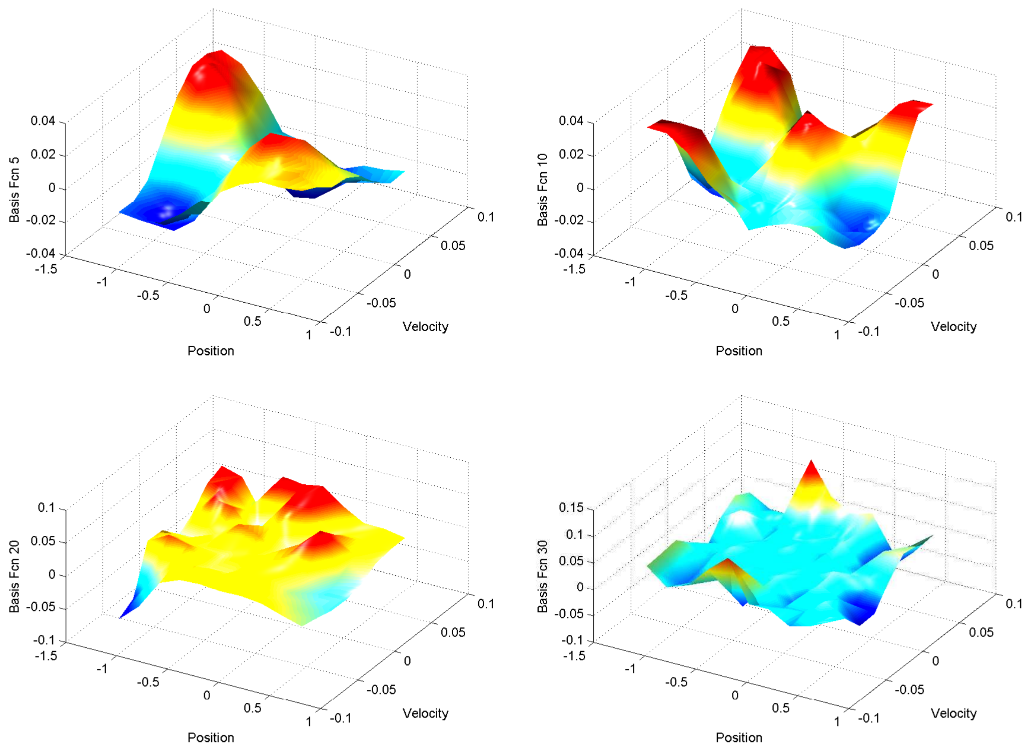


Figure 25: Proto-value functions learned by RPI on the mountain car task, showing (in row major order), the basis functions numbered 5, 10, 20, and 30 associated with the forward action.

The experiments reported below revealed significant differences between the two domains. In the mountain car domain, random subsampling produced significantly poorer results than the results shown earlier in Figure 24 using the trajectory subsampling method. In the inverted pendulum domain, performance with random subsampling was much better. One reason for this difference is the nature of the underlying manifold: the samples in the inverted pendulum are in a relatively narrow region around the 45 degree line. In contrast, the samples in the inverted pendulum domain are distributed across a wider region of the state space. Another difference in these domains was the nature of the random walk. In the inverted pendulum domain, the initial state was always set the same, with the pole starting from the vertical position at rest. In the mountain car domain, however, starting the car from a position of rest at the bottom of the hill produced poorer results than starting from the bottom with the velocities initialized randomly. Also, the experiments reported below used the raw state variables without any scaling. However, in the mountain car domain, scaling the velocity axis by a factor of 3 produces better results (in the results shown in Figure 24, the velocity dimension was scaled by a factor of 3). The effects of scaling are explored later in Figure 32. Finally, the performance of the RBF architecture was sensitive

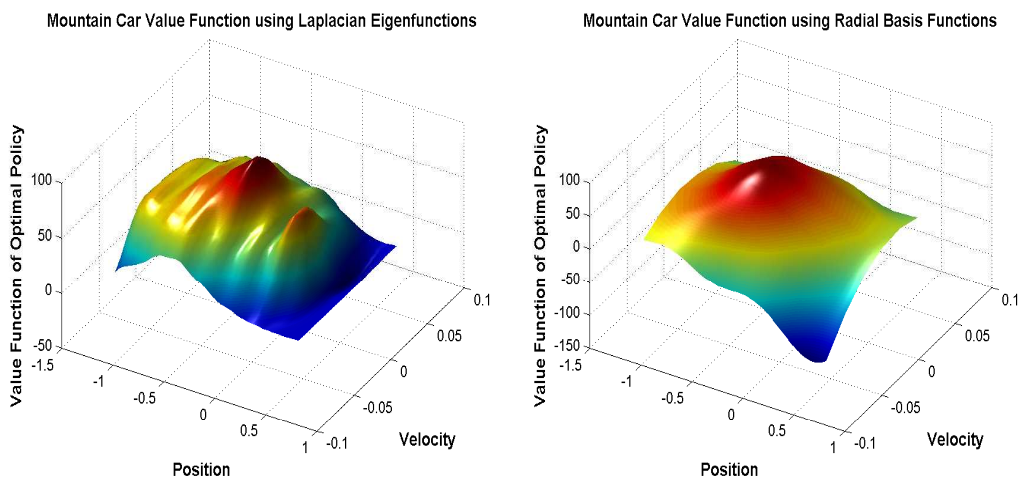


Figure 26: This figure compares the value function approximation using Laplacian eigenfunctions (left) vs. radial basis functions (right). Note the qualitative difference in the approximation.

to parameters such as width, which are not tuned below. The goal of this study was not to explore the performance of RBF, and hence we used a default radius of $\sigma = 1$.

Parameter	Inverted Pendulum	Mountain Car
T	(100 to 600)	(50 to 550)
N	500	90
Z	$\{100, \mathbf{500}, 1000\}$	$\{200, \mathbf{500}, 800, 1500\}$
ϵ	10^{-5}	10^{-3}
k	$\{25, \mathbf{50}, 100\}$	$\{15, \mathbf{25}, 40, \text{Gaussian}\}$
P	$\{25, \mathbf{50}, 100, 150\}$	$\{50, \mathbf{100}, 200\}$
δ	0.95	1
$\mathcal{O}$	varied	graph markov

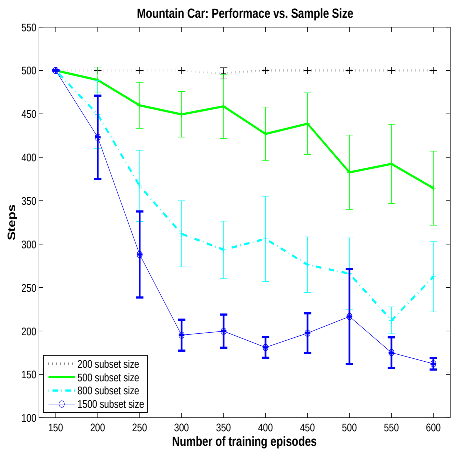
Table 4: Parameter values (as defined in Figure 21) for inverted pendulum and mountain car experiments. Default values are in bold.

While performance in both domains is measured by the number of steps, note that for the mountain car task, lower numbers indicate better performance since we are measuring the steps to reach the top of the hill. In the inverted pendulum, however, since we are measuring the number of steps that the pole remained upright, higher numbers indicate better performance. Figure 27 displays the first set of experiments, which varied the number of random samples $\mathcal{D}_{\mathcal{Z}}$ over which proto-value functions were computed. In the mountain car task, performance monotonically improves as the number of random samples is increased,

Number of RBFs	Inverted Pendulum RBF Parameters
10	3 x-axis, 3 y-axis, $\sigma = 1$
37	6 x-axis, 6 y-axis, $\sigma = 0.7$
50	7 x-axis, 7 y-axis, $\sigma = 0.3$
Number of RBFs	**Mountain Car RBF Parameters**
13	4 x-axis, 3 y-axis, $\sigma = 1$
49	8 x-axis, 6 y-axis, $\sigma = 1$

Table 5: RBF parameter settings for inverted pendulum and mountain car experiments.

up to a maximum of 1500 states. However, interestingly, in the pendulum task, a sample size of 500 produced the best results, and performance appears to degrade for larger sizes.



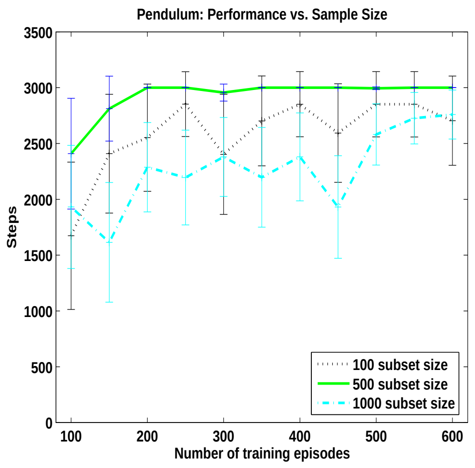
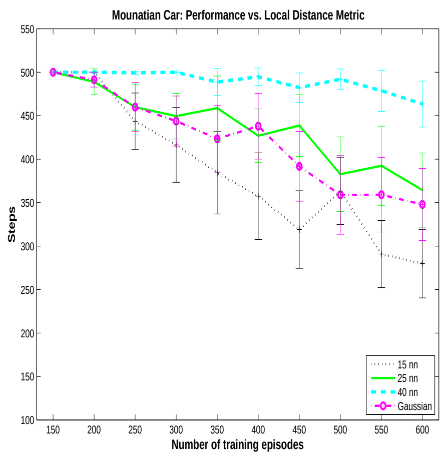


Figure 27: Performance on mountain car and inverted pendulum as a function of the number of subsampled points on which proto-value functions were computed.

In the second experiment, illustrated in Figure 28, the effect of varying the local distance metric was evaluated. In the mountain car domain, the nearest neighbor metric outperformed the gaussian distance metric. Also, using a lower number of nearest neighbors improved the performance. However, in the inverted pendulum task, performance improved as the number of nearest neighbors was increased, up to to 50, after which performance seemed to degrade.

Figure 29 varied the number of protovalue functions (PVFs) used. Here, there were significant differences in the two tasks. In the inverted pendulum task, performance dramatically improved from 25 to 50 proto-value functions, whereas in the mountain car domain, performance differences were less acute (note that the mountain car results used only 500 samples, which results in worse performance than using 1500 samples, as shown earlier in Figure 27).



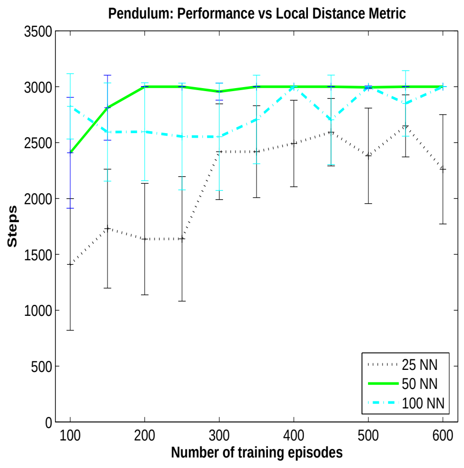
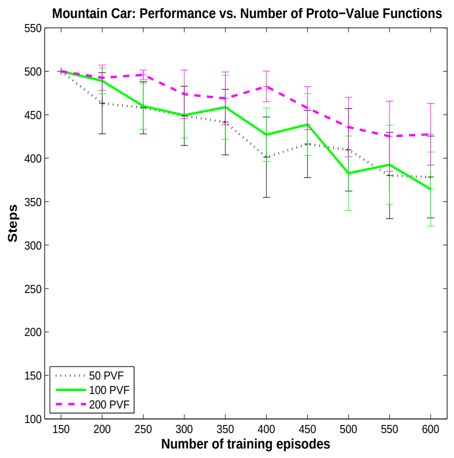


Figure 28: Performance on mountain car and inverted pendulum as a function of the nearest neighbors.



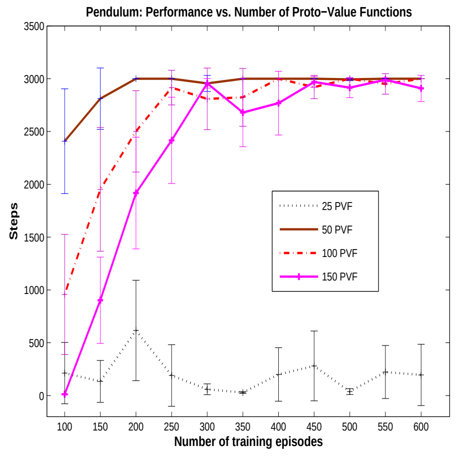


Figure 29: Performance on mountain car and inverted pendulum as a function of the number of proto-value functions.

Figure 30 measures the effect of varying the type of graph normalization in the pendulum task, which seems to cause little difference in the results. The same behavior was observed for the mountain car task (not shown).

Figure 31 compares the performance of the proto-value function with radial basis functions on the mountain car and inverted pendulum tasks. In the mountain car task, proto-value functions (bottom-most curve) significantly outperform several choices of RBFs (keeping the caveats mentioned above in mind). For the inverted pendulum, the performance of proto-value functions (top curve) appears significantly better than RBFs as well. These

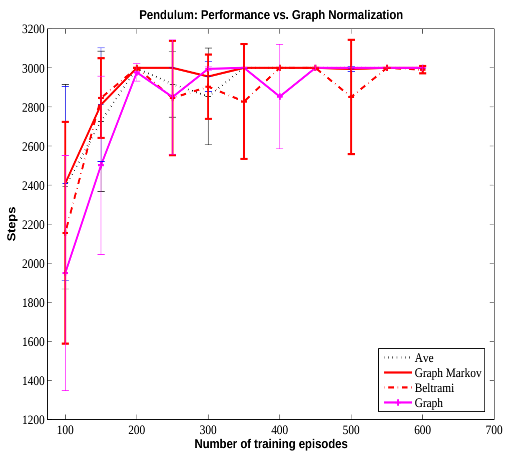


Figure 30: Results from the pendulum task, showing the variation in performance as a function of the type of graph normalization. Similar results were seen in the mountain car domain (not shown).

comparisons should be interpreted with some caution. The goal here is to provide a benchmark for comparison. In practice, the performance of RBFs and PVFs is determined by many factors, and it is possible for one to outperform the other depending on the parameters chosen (as the above sensitivity analysis clearly demonstrated). Overall, the experiments suggest that Laplacian PVFs can perform as well or better than some handcoded function approximators, indicating that they are a promising alternative to parametric function approximators. Ultimately, much more empirical experience will be needed before a full assessment of the Laplacian framework can be made.

In the last set of experiments, we analyze the sensitivity of Laplacian bases to scaling of the state space using the mountain car. In this task, the velocity dimension ranges between -0.07 to 0.07, whereas the position dimension ranges from -1.2 to 0.6, an order of magnitude larger. Figure 32 plots the performance of Laplacian PVFs (average number of steps to goal) over 20 learning runs, for several different choices of scaling. In each experiment, the velocity dimension was scaled by some number, ranging from 1 (no scaling) to 4. As the results show, there is some sensitivity to scaling, with the best results obtained for a scaling factor of 3. The results shown in this figure were obtained using trajectory sampling.

10. Discussion

Many extensions of the framework proposed in this paper are being actively explored, which are briefly summarized here. We have naturally restricted our discussion to the simplest methods, but the scope of proto-value functions can easily be extended to cover more general situations as discussed below.

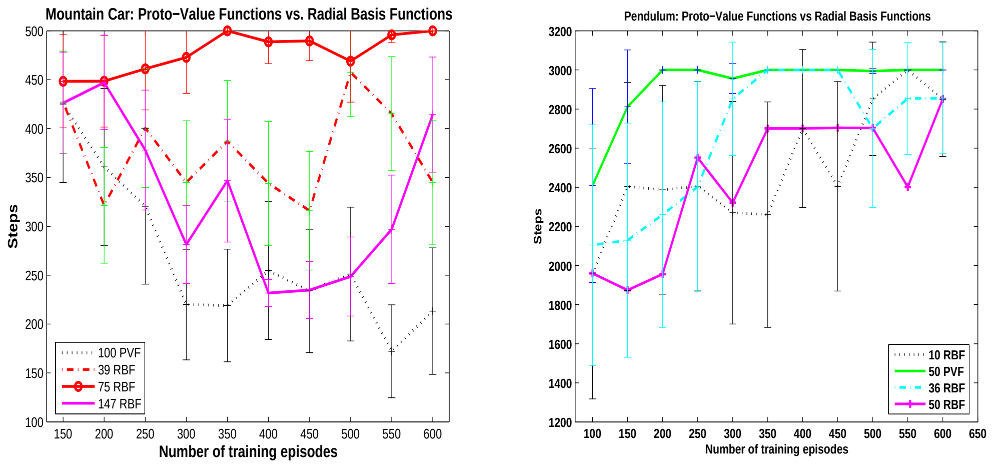


Figure 31: A comparison of radial basis functions and proto-value functions on the mountain car task (left) and the inverted pendulum task (right).

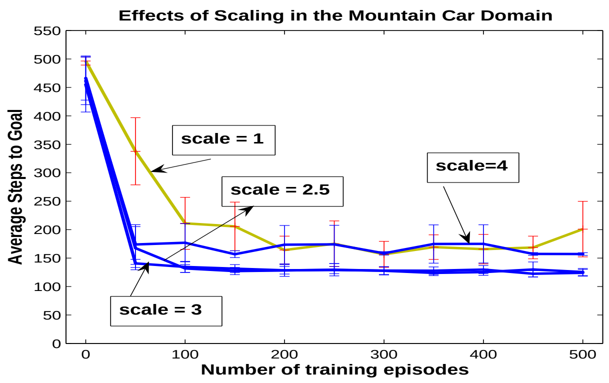


Figure 32: This experiment investigates the effect of scaling the velocity axis in the mountain car domain on the performance of the Laplacian bases. The results show that the best performance is achieved by scaling the velocity dimension by a factor of 3. These results were obtained using trajectory subsampling to build the Laplacian bases. The results shown used 30 basis functions for each action.

10.1 Bias-Variance Analysis

A fundamental property of any machine learning framework is its *bias-variance* tradeoff (Hastie et al., 2001). More precisely, the mean-squared error of any approximator can be decomposed into a sum of two terms: a *bias* term reflecting how far the average prediction differs from the true prediction; and a *variance* term reflecting how far each prediction differs from the average prediction. Parametric function approximators of low-dimension, such as polynomials of low degree or RBFs with a low number of basis functions, are at one end of the bias-variance spectrum, constituting an approximation architecture whose bias is high (since the true value may be far from the approximated value), but whose variance is low (since each set of predictions on a random dataset may not be too far off from the averaged set of predictions). The proposed Laplacian framework can be similarly characterized at the other extreme: it provides a highly flexible approximation architecture with customizable basis functions, with potentially low bias (at least for sufficient numbers of basis functions), but whose variance may be high for low to medium number of samples. An important ongoing research direction is to make precise this bias-variance tradeoff. It is important to emphasize there are a variety of intermediate approximation architectures that lie in between pure graph-based methods and pure parametric methods, whose bias-variance tradeoff on some problems may be more desirable.

10.2 Incremental Methods

A natural question is whether representation learning by spectral analysis of the graph Laplacian can be done incrementally, making it easier to combine with incremental parameter estimation techniques such as Q-learning (Watkins, 1989). Decentralized spectral algorithms for computing the top k eigenvectors of a symmetric matrix have been developed recently (Kempe and McSherry, 2004), which rely on *gossip* algorithms for computing averages over values stored at the nodes of a graph. In these algorithms, the amount of computation required at each node is $\mathcal{O}(k^3)$ for computing the top k eigenvectors, which can be substantially smaller than centralized methods which require $\mathcal{O}(kn^2)$ (here n is the number of vertices of the graph). Furthermore, the adjacency matrix of the graph is never stored centrally. An interesting problem for future research is to develop incremental methods for computing proto-value functions using gossip algorithms, and combine them with incremental RL methods, such as Q-learning. An incremental ISOMAP manifold learning technique has also recently been developed (Law and Jain, 2006).

10.3 Directed Graphs and Non-reversible Random Walks

We have restricted the analysis and construction of proto-value functions to "off-policy" basis functions, where the random walk on an undirected graph is used to produce the basis representations. Of necessity, this random walk represents a reversible Markov process. It is possible to extend our approach to both "on-policy" analysis by directly diagonalizing the *Green's function* associated with a policy (see (Maggioni and Mahadevan, 2005) and the second paper (Maggioni and Mahadevan, 2006) for more details). In particular, we have recently implemented the directed graph Laplacian proposed by Chung (2005)), which is defined as

$$L_D = D_\phi - \frac{D_\phi P + P^T D_\phi}{2}$$

where D_ϕ is a diagonal matrix whose entries are given by $\phi(v)$, the *Perron* vector or leading eigenvector associated with the spectral radius of the transition matrix P specifying the directed random walk on G. For a strongly connected directed graph G, the Perron-Frobenius theorem can be applied to show that the transition matrix is irreducible and non-negative, and consquently the leading eigenvector associated with the largest (real) eigenvalue must have all positive components $\phi(v) > 0$. Our initial results (Johns et al., 2006) using the directed graph Laplacian above indicate that in some cases, it can result in more efficient approximations in MDPs when actions are highly directional (e.g., a modifed two-room task where there are two "one-way" doors leading from one room to the other).

10.4 Theoretical Analysis

Another ongoing research direction is providing theoretical guarantees on the efficiency of proto-value functions in approximating value functions. As discussed in Section 5, some results follow immediately from the construction of proto-value functions. For example, it can be shown easily that the approximation produced by projecting a given function on a graph to the smallest k proto-value functions produces *globally* the smoothest approximation. There are also classical results on the efficiency of Fourier bases for approximating smooth functions in a Sobolev space (Mallat, 1989), which can be carried over to the discrete case of graphs. We discuss some of these issues at length in the companion paper (Maggioni and Mahadevan, 2006). Another direction is to study the convergence of RPI, which must of necessity sample the underlying manifold on the state space. In this case, the theoretical analysis must depend on making some assumptions regarding the underlying manifold. In the interests of generality and ease of readability, we have ignored these important theoretical considerations. Belkin and Niyogi (2005) show that under uniform sampling conditions, the graph Laplacian (constructed in a certain way) converges to the Laplace-Beltrami operator on the underlying manifold. However, it may be difficult to guarantee uniform sampling in the reinforcement learning context, where an agent has to choose actions to explore its environment. We are also currently exploring the stability of the subspaces defined by proto-value functions using the tools of matrix perturbation theory (Stewart and Sun, 1990), which quantifies the degree to which small perturbations of (positive definite) matrices lead to bounded changes in the spectrum and eigenspace as well. Such analyses have been carried out for eigenvector methods, such as PageRank (Ng et al., 2001b), and would provide a deeper insight into the robustness of spectral methods for value function approximation.

10.5 Proto-Value Functions for Hierarchical Reinforcement Learning

As mentioned in the introduction, proto-value functions provide a broad theoretical framework that unifies the study of problems raised in recent work on automating hierarchical reinforcement learning (Barto and Mahadevan, 2003). These include the question of decomposing the overall state space by finding bottlenecks (Hengst, 2002) and symmetries (Ravindran and Barto, 2003). For example, as shown earlier in Figure 3, the second proto-value function can be thresholded to find bottlenecks. In the second paper (Maggioni and Mahadevan, 2006), the multiresolution diffusion wavelet method provides an even more powerful method for constructing sparse representations of temporally extended actions.

10.6 Transfer Across Tasks

Proto-value functions are learned not from rewards, but from the topology of the underlying state space (in the "off-policy" case). Consequently, they immediately suggest a solution to the well-known problem of transfer in reinforcement learning, which has been studied by many researchers (e.g., (Mahadevan, 1992; Sherstov and Stone, 2005)). One key advantage of proto-value functions is that they provide a theoretically justified framework for transfer, which respects the underlying manifold. We have recently begun a new framework called *proto-transfer* learning, which explores the transfer of learned representations from one task to another (in contrast to transferring learned policies) (Ferguson and Mahadevan, 2006).

11. Summary

This paper describes a unified framework for learning representation and behavior in Markov decision processes. The fundamental novel concept is the notion of a proto-value function, which represent building blocks of the set of all (square-integrable) value functions on a manifold or graph. Proto-value functions can be defined in several ways: this first paper focuses on the eigenfunctions of the graph Laplacian which formally constitutes a global Fourier analysis of the state space manifold. Eigenfunctions were shown to have properties crucial for value function approximation: projections of a function onto the eigenfunctions of the graph Laplacian not only provide the globally smoothest approximation, but also results in an approximation that respects the underlying manifold. An immediate consequence of this approach is that nonlinearities and other global geometric invariants are automatically captured by proto-value functions. A novel algorithm called representation policy iteration (RPI) was described that was able to use proto-value functions to learn optimal policies in Markov decision processes. Several variants of RPI were described, including ones for large factored MDPs as well as for continuous MDPs. The extension to continuous states was accomplished using the Nystrom̈ interpolation method. Detailed experimental results from discrete and continuous domains not only showed the effectiveness of the proto-value function approach, but also provided some evidence that this approach may outperform handcoded parametric function approximators such as polynomials and radial basis functions. Much research remains to be done in this framework: as such, this paper represents a first step on the road to a new unified paradigm combining the learning of representation and behavior.

Appendix: Implementation Details

The experiments reported in this paper were implemented in MATLAB. The eigenfunctions of the Laplacian were computed using the MATLAB `eigs` routine. Since the graphs constructed for both discrete and continuous MDPs were very sparse, much of the code was optimized to exploit sparse matrix operations.The control learning component of RPI was implemented using a highly modified MATLAB implementation of LSPI. We are indebted to Michail Lagoudakis and Ronald Parr for making available their MATLAB software for LSPI. The modifications to LSPI include vectorized optimizations, batch methods for computing basis functions, and caching of intermediate results wherever possible.

We have refrained from giving CPU times in the paper, since running times will invariably be platform specific. For example, on a fast Intel 2 Gigahertz dual-core laptop running Windows XP, RPI learns to solve the inverted pendulum task in around 5 seconds (this includes basis function construction as well as least-squares fixpoint updates). The mountain car task takes about 20 seconds. In the blockers task, RPI took around a minute to learn a good policy in the 10×10 grid with 3 agents and 3 blockers. Further code optimizations in progress will improve these times. We intend to distribute the RPI codebase, including several discrete and continuous testbeds, and the Laplacian eigenfunction and the diffusion wavelet basis software, using the RL repository.

Acknowledgments

We would like to acknowledge the help and support of many researchers who have inspired and helped us during the course of this research. We thank Kimberly Ferguson and Sarah Osentoski for carrying out the experiments on the inverted pendulum and the mountain car domains. We would like to thank Andrew Barto, and other current and former members of the Autonomous Learning Laboratory, including Ravi Balaraman, Mohammad Ghavamzadeh, Jeff Johns, George Konidaris, Amy McGovern, Khashayar Rohanimanesh, and Ozgur Simsek for their feedback. Support for this research was provided in part by the National Science Foundation under grants ECS-0218125, IIS-0534999 and DMS-0512050.

References

S. Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier/North-Holland, 1968.

S. Axler, P. Bourdon, and W. Ramey. *Harmonic Function Theory.* Springer, 2001.

J. Bagnell and J. Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

C T H Baker. *The numerical treatment of integral equations.* Oxford: Clarendon Press, 1977.

A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems Journal*, 13:41–77, 2003.

M. Belkin and P. Niyogi. Towards a theoretical foundation for laplacian-based manifold methods. In *Proceedings of the International Conference on Computational Learning Theory*, 2005.

M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds. *Machine Learning*, 56:209–239, 2004.

S Belongie, C Fowlkes, F Chung, and J Malik. Spectral partitioning with indefinite kernels using the Nyström extension. *ECCV*, 2002.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

J. A. Boyan. Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

James C Bremer, Ronald R Coifman, Mauro Maggioni, and Arthur D Szlam. Diffusion wavelet packets. *Tech. Rep. YALE/DCS/TR-1304, Yale Univ., Appl. Comp. Harm. Anal., submitted*, Sep. 2004. doi: http://www.math.yale.edu/$\sim$mmm82/DiffusionWaveletPackets.pdf.

J Cheeger. A lower bound for the smallest eigenvalue of the laplacian. In RC Gunning, editor, *Problems in Analysis*, pages 195–199. Princeton Univ. Press, 1970.

T. Chow. The Q-spectrum and spanning trees of tensor products of bipartite graphs. *Proceedings of the American Mathematical Society*, 125(11):3155–3161, 1997.

F Chung. Laplacians and the Cheeger Inequality for Directed Graphs. *Annals of Combinatorics*, 2005.

Fan Chung. *Spectral Graph Theory.* Number 92. CBMS-AMS, May 1997.

Ronald R Coifman and Mauro Maggioni. Diffusion wavelets. *Tech. Rep. YALE/DCS/TR-1303, Yale Univ., Appl. Comp. Harm. Anal.*, Sep. 2004. doi: http://www.math.yale.edu/$\sim$mmm82/DiffusionWavelets.pdf. to appear.

Ronald R Coifman, Yves Meyer, S Quake, and Mladen V Wickerhauser. Signal processing and compression with wavelet packets. In *Progress in wavelet analysis and applications (Toulouse, 1992)*, pages 77–93. Frontières, Gif, 1993.

Ronald R Coifman, Stephane Lafon, Ann Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data. part i: Diffusion maps. *Proc. of Nat. Acad. Sci.*, (102): 7426–7431, May 2005a.

Ronald R Coifman, Stephane Lafon, Ann Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data. part ii: Multiscale methods. *Proc. of Nat. Acad. Sci.*, (102): 7432–7438, May 2005b.

Ronald R Coifman, Mauro Maggioni, Steven W Zucker, and Iannis G Kevrekidis. Geometric diffusions for the analysis of data from sensor networks. *Curr Opin Neurobiol*, 15(5):576–84, October 2005c.

D. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs: Theory and Application.* Academic Press, 1980.

I Daubechies. *Ten lectures on wavelets.* Society for Industrial and Applied Mathematics, 1992. ISBN 0-89871-274-2.

P. Dayan. Improving generalisation for temporal difference learning: The successor representation. *Neural Computation*, 5:613–624, 1993.

Frank Deutsch. *Best Approximation In Inner Product Spaces.* Canadian Mathematical Society, 2001.

T. Dietterich and X. Wang. Batch value function approximation using support vectors. In *Proceedings of Neural Information Processing Systems*. MIT Press, 2002.

P Drineas and M W Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *J. Machine Learning Research*, (6):2153–2175, 2005.

P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. Technical Report YALEU/DCS/TR-1270, Yale University Department of Computer Science, New Haven, CT, February 2004.

C. Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of AI Research*, 16:59–104, 2002.

D. Farias. The linear programming approach to approximate dynamic programming. In *Learning and Approximate Dynamic Programming: Scaling up to the Real World*. John Wiley and Sons, 2003.

K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Transfer Learning*, 2006.

M. Fiedler. Algebraic connectivity of graphs. *Czech. Math. Journal*, 23(98):298–305, 1973.

D. Foster and P. Dayan. Structure in the space of value functions. *Machine Learning*, 49: 325–346, 2002.

A Frieze, R Kannan, and S Vempala. Fast Monte Carlo algorithms for finding low-rank approximations. In *Proceedings of the 39th annual IEEE symposium on foundations of computer science*, pages 370–378, 1998.

G. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Department of Computer Science, Carnegie Mellon University, 1995.

C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored markov decision processes. In *Proceedings of the 15th IJCAI*, 2001.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of AI Research*, 19:399–468, 2003.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *ICML*, pages 243–250, 2002.

J. Johns, S. Osentoski, and S. Mahadevan. Approximating action value functions using the directed graph laplacian, 2006. Submitted.

S. Kakade. A natural policy gradient. In *Proceedings of Neural Information Processing Systems*. MIT Press, 2002.

D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 561–568. Morgan Kaufmann Publishers, San Francisco, US, 2004.

D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proceedings of the 16th Conference on Uncertainty in AI*, 2000.

R. Kondor and R. Vert. Diffusion kernels. In *Kernel Methods in Computational Biology*. MIT Press, 2004.

R. M. Kretchmar and C. W. Anderson. Using temporal neighborhoods to adapt function approximators in reinforcement learning. In *International Work Conference on Artificial and Natural Neural Networks*, 1999.

P. Perona L. Zelnik-Manor. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, 2004.

J. Lafferty and G. Lebanon. Diffusion kernels on statistical manifolds. *Journal of Machine Learning Research*, 6:129–163, 2005.

Stephane Lafon. *Diffusion maps and geometric harmonics*. PhD thesis, Yale University, Dept of Mathematics & Applied Mathematics, 2004.

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

J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Press, 1991.

S. Lavalle. Planning algorithms, 2005. Forthcoming text, to appear.

M. Law and A. Jain. Incremental nonlinear dimensionality reduction by manifold learning. *IEEE PAMI*, 2006.

J. M. Lee. *Introduction to Smooth Manifolds*. Springer, 2003.

M. Maggioni and S. Mahadevan. A multiscale framework for markov decision processes using diffusion wavelets. submitted, 2006.

M. Maggioni and S. Mahadevan. Fast direct policy evaluation using multiscale analysis of markov diffusion processes. In *University of Massachusetts, Department of Computer Science Technical Report TR-2005-39*, 2005.

S. Mahadevan. Samuel Meets Amarel: Automating Value Function Approximation using Global State Space Analysis. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, 2005a. AAAI Press/MIT Press.

S. Mahadevan. Proto-Value Functions: Developmental Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, 2005b.

S. Mahadevan. Enhancing transfer in reinforcement learning by building stochastic models of robot actions. In *Proceedings of the Ninth International Conference on Machine Learning, Aberdeen, Scotland*, pages 290–299, 1992.

S. Mahadevan. Representation Policy Iteration. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005c.

S. Mahadevan, M. Maggioni, Kimberly Ferguson, and Sarah Osentoski. Learning representation and control in continuous markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

Stephane Mallat. *A wavelet tour in signal processing.* Academic Press, 1998.

Stephane G Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):674–693, 1989. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.192463.

Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *ICML*, 2004.

A. McGovern. *Autonomous Discovery of Temporal Abstractions from Interactions with an Environment.* PhD thesis, University of Massachusetts, Amherst, 2002.

N. Menache, N. Shimkin, and S. Mannor. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134:215–238, 2005.

R. Munos. Error bounds for approximate value iteration. In *AAAI*, 2005.

R. Munos. Error bounds for approximate policy iteration. In *ICML*, 2003.

A. Nedic and D. Bertsekas. Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Systems Journal*, 13, 2003.

A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm, 2001a. URL `citeseer.ist.psu.edu/ng01spectral.html`.

A. Ng, A. Zheng, and M. Jordan. Link analysis, eigenvectors, and stability. In *Proceedings of the 15th IJCAI*, 2001b.

A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, 2002.

Partha Niyogi, I Matveeva, and Mikhail Belkin. Regression and regularization on large graphs. Technical report, University of Chicago, Nov. 2003.

D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3): 161–178, 2002.

J. Peters, S. Vijaykumar, and S. Schaal. Reinforcement learning for humanoid robots. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

J C Platt. FastMap, MetricMap, and Landmark MDS are all Nyström algorithms. Technical Report MSR-TR-2004-26, Microsoft Research, Sep 2004.

P. Poupart, R. Patrascu, D. Schuurmans, C. Boutilier, and C. Guestrin. Greedy linear value function approximation for factored markov decision processes. In *AAAI*, 2002.

M. L. Puterman. *Markov decision processes*. Wiley Interscience, New York, USA, 1994.

C. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Proceedings of the International Conference on Neural Information Processing Systems*. MIT Press, 2004.

B. Ravindran and A. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In *Proceedings of the 18th IJCAI*, 2003.

S Rosenberg. *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1997.

S. Roweis and L. Saul. Nonlinear dimensionality reduction by local linear embedding. *Science*, 290:2323–2326, 2000.

B. Sallans and G. Hinton. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5:1063–1088, 2004.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

A. Sherstov and P. Stone. Improving action selection in mdp's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE PAMI*, 22:888–905, 2000.

Özgür Simsek and Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML*, 2004.

G. Stewart and J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

D. Subramanian. *A theory of justified reformulations*. Ph.D. Thesis, Stanford University, 1989.

R. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, 1984.

R. Sutton and A. G. Barto. *An Introduction to Reinforcement Learning*. MIT Press, 1998.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988. URL `citeseer.ist.psu.edu/sutton88learning.html`.

J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.

J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

P. Utgoff and D. Stracuzzi. Many-layered learning. *Neural Computation*, 14:2497–2529, 2002.

C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, pages 682–688, 2000.

X. Zhou. *Semi-Supervised Learning With Graphs*. PhD thesis, Carnegie Mellon University, 2005.