Journal of Machine Learning Research 8 (2007) 2169-2231 Submitted 6/06; Revised 3/07; Published 9/07

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@DUKE.EDU
Department of Mathematics and Computer Science
Duke University
Durham, NC 27708

Editor: Carlos Guestrin

Abstract

This paper introduces a novel *spectral* framework for solving Markov decision processes (MDPs) by jointly learning representations and optimal policies. The major components of the framework described in this paper include: (i) A general scheme for constructing representations or *basis functions* by diagonalizing symmetric *diffusion* operators (ii) A specific instantiation of this approach where global basis functions called *proto-value functions* (PVFs) are formed using the eigenvectors of the *graph Laplacian* on an undirected graph formed from state transitions induced by the MDP (iii) A three-phased procedure called *representation policy iteration* comprising of a sample collection phase, a representation learning phase that constructs basis functions from samples, and a final parameter estimation phase that determines an (approximately) optimal policy within the (linear) subspace spanned by the (current) basis functions. (iv) A specific instantiation of the RPI framework using least-squares policy iteration (LSPI) as the parameter estimation method (v) Several strategies for scaling the proposed approach to large discrete and continuous state spaces, including the *Nyström* extension for out-of-sample interpolation of eigenfunctions, and the use of *Kronecker sum factorization* to construct compact eigenfunctions in product spaces such as factored MDPs (vi) Finally, a series of illustrative discrete and continuous control tasks, which both illustrate the concepts and provide a benchmark for evaluating the proposed approach. Many challenges remain to be addressed in scaling the proposed framework to large MDPs, and several elaboration of the proposed framework are briefly summarized at the end.

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

1. Introduction

This paper introduces a novel *spectral* framework for solving Markov decision processes (MDPs) (Puterman, 1994) where both the underlying representation or basis functions and (approximate) optimal policies within the (linear) span of these basis functions are simultaneously learned. This framework addresses a major open problem not addressed by much previous work in the field of *approximate dynamic programming* (Bertsekas and

Tsitsiklis, 1996) and *reinforcement learning* (Sutton and Barto, 1998), where the set of "features" or *basis functions* mapping a state s to a k-dimensional real vector $\phi(s) \in \mathbb{R}^k$ is usually hand-engineered.

The overall framework can be summarized briefly as follows. The underlying task environment is modeled as an MDP, where the system dynamics and reward function are typically assumed to be unknown. An agent explores the underlying state space by carrying out actions using some policy, say a random walk. Central to the proposed framework is the notion of a *diffusion model* (Coifman et al., 2005a; Kondor and Lafferty, 2002): the agent constructs a (directed or undirected) graph connecting states that are "nearby". In the simplest setting, the diffusion model is defined by the *combinatorial graph Laplacian* matrix $L = D - W$, where W is a symmetrized weight matrix, and D is a diagonal matrix whose entries are the row sums of W.[1] Basis functions are derived by diagonalizing the Laplacian matrix L, specifically by finding its "smoothest" eigenvectors that correspond to the smallest eigenvalues. Eigenvectors capture large-scale temporal properties of a transition process. In this sense, they are similar to value functions, which reflect the accumulation of rewards over the long run. The similarity between value functions and the eigenvectors of the graph Laplacian sometimes can be remarkable, leading to a highly compact encoding (measured in terms of the number of basis functions needed to encode a value function). Laplacian basis functions can be used in conjunction with a standard "black box" parameter estimation method, such as Q-learning (Watkins, 1989) or least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) to find the best policy representable within the space of the chosen basis functions.

While the overall goal of learning representations is not new within the context of MDPs—it has been addressed by Dayan (1993) and Drummond (2002) among others—our approach is substantially different from previous work. The fundamental idea is to construct basis functions for solving MDPs by diagonalizing symmetric diffusion operators on an empirically learned graph representing the underlying state space. A diffusion model is intended to capture information flow on a graph or a *manifold*.[2] A simple diffusion model is a random walk on an undirected graph, where the probability of transitioning from a vertex (state) to its neighbor is proportional to its degree, that is $P_r = D^{-1}W$ (Chung, 1997). As we will see in Section 3, the combinatorial Laplacian operator L defined in the previous paragraph is closely related spectrally to the random walk operator P_r. A key advantage of diffusion models is their simplicity: it can be significantly easier to estimate a "weak" diffusion model, such as the undirected random walk P_r or the combinatorial Laplacian L, than to learn the true underlying transition matrix P^π of a policy π.

The proposed framework can be viewed as automatically generating subspaces on which to project the value function using spectral analysis of operators on graphs. This differs fundamentally from many past attempts at basis function generation, for example tuning the parameters of pre-defined basis functions (Menache et al., 2005; Kveton and Hauskrecht,

1. Section 9 describes how to generalize this simple diffusion model in several ways, including *directed* graphs where the symmetrization is based on the *Perron* vector or the leading eigenvector associated with the largest eigenvalue (Chung, 2005), state-action diffusion models where the vertices represent state-action pairs, and diffusion models for temporally extended actions.
2. Intuitively, a manifold is a (finite or infinite) set that looks "locally Euclidean", in that an invertible mapping can be defined from a neighborhood around each element of the set to $\mathbb{R}^n$. There are technical conditions that additionally need to be satisfied for a manifold to be *smooth*, as explained in Lee (2003).

2006), dynamically allocating new parametric basis functions based on state space trajectories (Kretchmar and Anderson, 1999), or generating basis functions using the Bellman error in approximating a specific value function (Keller et al., 2006; Petrik, 2007; Parr et al., 2007; Patrascu et al., 2002). The main contribution of this paper is to show how to construct *novel* non-parametric basis functions whose shapes reflect the geometry of the environment.

In this paper, basis functions are constructed from spectral analysis of diffusion operators where the resulting representations are constructed without explicitly taking rewards into account. This approach can be contrasted with recent approaches that explicitly use reward information to generate basis functions (Keller et al., 2006; Petrik, 2007; Parr et al., 2007). There are clear advantages and disadvantages to these two approaches. In the non-reward based approach, basis functions can be more easily transferred across MDPs in situations where an agent is required to solve multiple tasks defined on a common state (action) space. Furthermore, basis functions constructed using spectral analysis reflect global geometric properties, such as bottlenecks and symmetries in state spaces, that are invariant across multiple MDPs on the same state (action) space. Finally, in the full control learning setting studied here, the agent does not initially know the true reward function or transition dynamics, and building representations based on estimating these quantities introduces another potential source of error. It is also nontrivial to learn accurate transition models, particularly in continuous MDPs. However, in other settings such as planning, where the agent can be assumed to have a completely accurate model, it is entirely natural and indeed beneficial to exploit reward or transition dynamics in constructing basis functions. In particular, it is possible to design algorithms for basis function generation with provable performance guarantees (Parr et al., 2007; Petrik, 2007), although these theoretical results are at present applicable only to the more limited case of evaluating a *fixed* policy. The proposed framework can in fact be easily extended to use reward information by building reward-based diffusion models, as will be discussed in more detail in Section 9.

Since eigenvectors of the graph Laplacian form "global" basis functions whose support is the entire state space, each eigenvector induces a real-valued mapping over the state space. Consequently, we can view each eigenvector as a "proto-value" function (or PVF), and the set of PVFs form the "building blocks" of all value functions on a state space (Mahadevan, 2005a). Of course, it is easy to construct a complete orthonormal set of basis functions spanning all value functions on a graph: the unit vectors themselves form such a basis, and indeed, any collection of $|S|$ random vectors can (with high probability) be orthonormalized so that they are of unit length and "perpendicular" to each other. The challenge is to construct a *compact* basis set that is *efficient* at representing value functions with as *few* basis functions as possible. Proto-value functions differ from these obvious choices, or indeed other more common parametric choices such as radial basis functions (RBFs), polynomial bases, or CMAC, in that they are associated with the spectrum of the Laplacian which has an intimate relationship to the large-scale geometry of a state space. The eigenvectors of the Laplacian also provide a *systematic* organization of the space of functions on a graph, with the "smoothest" eigenvectors corresponding to the smallest eigenvalues (beginning with the constant function associated with the zero eigenvalue). By projecting a given value function on the space spanned by the eigenvectors of the graph Laplacian, the "spatial" content of a value function is mapped into a "frequency" basis, a hallmark of classical "Fourier" analysis (Mallat, 1989).

It has long been recognized that traditional parametric function approximators may have difficulty accurately modeling value functions due to nonlinearities in an MDP's state space (Dayan, 1993). Figure 1 illustrates the problem with a simple example.[3] In particular, as Dayan (1993) and Drummond (2002) among others have noted, states close in Euclidean distance may have values that are very far apart (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, these approaches have lacked a sufficiently comprehensive and broad theoretical framework (related work is discussed in more detail in Section 8). We show that by rigorously formulating the problem of value function approximation as *approximating real-valued functions on a graph or manifold using a diffusion model*, a more general solution emerges that not only has broader applicability than these previous methods, but also enables a novel framework called *Representation Policy Iteration* (RPI) (Mahadevan, 2005b) where representation learning is interleaved with policy learning. The RPI framework consists of an outer loop that learns basis functions from sample trajectories, and an inner loop that consists of a control learner that finds improved policies.

Figure 2 shows a set of samples produced by doing a random walk in the inverted pendulum task. In many continuous control tasks, there are often physical constraints that limit the "degrees of freedom" to a lower-dimensional manifold, resulting in motion along highly constrained regions of the state space. Instead of placing basis functions uniformly in all regions of the state space, the proposed framework recovers the underlying manifold by building a graph based on the samples collected over a period of exploratory activity. The basis functions are then computed by diagonalizing a diffusion operator (the Laplacian) on the space of functions on the graph, and are thereby customized to the manifold represented by the state (action) space of a particular control task. In discrete MDPs, such as Figure 1, the problem is one of compressing the space of (value) functions $\mathbb{R}^{|S|}$ (or $\mathbb{R}^{|S|\times|A|}$ for action-value functions). In continuous MDPs, such as Figure 2, the corresponding problem is compressing the space of square-integrable functions on $\mathbb{R}^2$, denoted as $\mathbb{L}^2(\mathbb{R}^2)$. In short, the problem is one of dimensionality reduction not in the data space, but on the space of functions on the data.[4]

Both the discrete MDP shown in Figure 1 and the continuous MDP shown in Figure 2 have "inaccessible" regions of the state space, which can be exploited in focusing the function approximator to accessible regions. Parametric approximators, as typically constructed, do not distinguish between accessible and inaccessible regions. Our approach goes beyond modeling just the reachable state space, in that it also models the local *non-uniformity* of a given region. This non-uniform modeling of the state space is facilitated by constructing a graph operator which models the local density across regions. By constructing basis functions adapted to the non-uniform density and geometry of the state space, our approach extracts significant topological information from trajectories. These ideas are formalized in Section 3.

The additional power obtained from knowledge of the underlying state space graph or manifold comes at a potentially significant cost: the manifold representation needs to be

3. Further details of this environment and similar variants are given in Section 2.1 and Section 4.2.
4. The graph Laplacian induces a smoothing prior on the space of functions of a graph that can formally be shown to define a data-dependent reproducing kernel Hilbert space (Scholkopf and Smola, 2001).

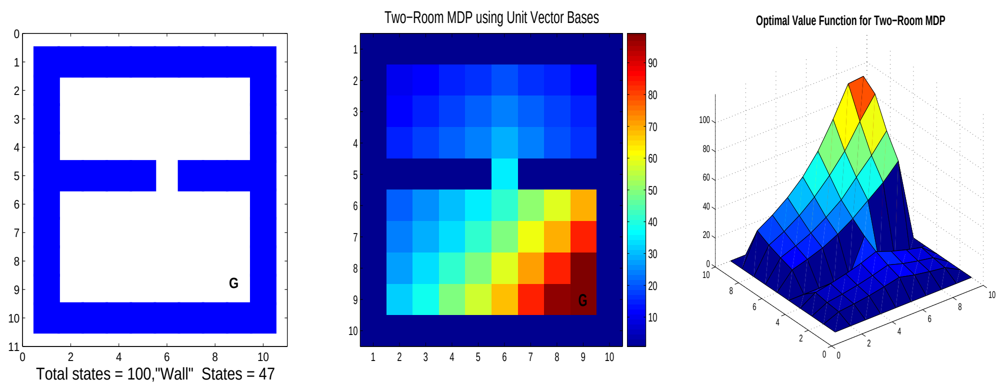


Figure 1: It is difficult to approximate nonlinear value functions using traditional parametric function approximators. Left: a “two-room” environment with 100 total states, divided into 57 accessible states (including one doorway state), and 43 inaccessible states representing exterior and interior walls (which are “one state” thick). Middle: a $2D$ view of the optimal value function for the two-room grid MDP, where the agent is rewarded for reaching the state marked G and its immediate neighbors by $+100$. Access to each room from the other is only available through a central door, and this “bottleneck” results in a strongly nonlinear optimal value function. Right: a $3D$ plot of the optimal value function, where the axes are reversed for clarity.

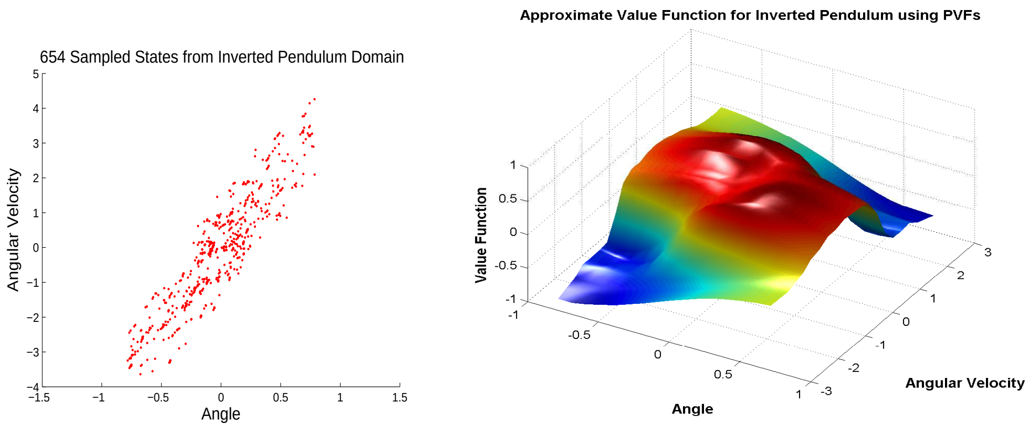


Figure 2: Left: Samples from a series of random walks in an inverted pendulum task. Due to physical constraints, the samples are largely confined to a narrow region. The proto-value function framework presented in this paper empirically models the underlying manifold in such continuous control tasks, and derives customized basis functions that exploit the unique structure of such point-sets in $\mathbb{R}^n$. Right: An approximation of the value function learned by using PVFs.

learned, and furthermore, basis functions need to be computed from it. Although our paper demonstrates that eigenvector-type basis functions resulting from a diffusion analysis of graph-based manifolds can solve standard benchmark discrete and continuous MDPs, the problem of efficiently learning manifold representations of *arbitrary* MDPs is beyond the scope of this introductory paper. We discuss a number of outstanding research questions in Section 9 that need to be addressed in order to develop a more complete solution.

One hallmark of Fourier analysis is that the basis functions are localized in frequency, but not in time (or space). Hence, 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: can Laplacian bases be computed and represented compactly in large discrete and continuous spaces?[5] We will address this problem in particular cases of interest: 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 *Nyström* interpolation method, Laplacian bases can be constructed quite efficiently (Drineas and Mahoney, 2005). Finally, a variety of other techniques can be used to sparsify Laplacian bases, including graph partitioning (Karypis and Kumar, 1999), matrix sparsification (Achlioptas et al., 2002), and automatic Kronecker matrix factorization (Van Loan and Pitsianis, 1993). Other sources of information can be additionally exploited to facilitate sparse basis construction. For example, work on hierarchical reinforcement learning surveyed in Barto and Mahadevan (2003) studies special types of MDPs called semi-Markov decision processes, where actions are temporally extended, and value functions are decomposed using the hierarchical structure of a task. In Section 9, we discuss how to exploit such additional knowledge in the construction of basis functions.

This research is related to recent work on manifold and spectral learning (Belkin and Niyogi, 2004; Coifman and Maggioni, 2006; Roweis and Saul, 2000; Tenenbaum et al., 2000). A major difference is that our focus is on solving Markov decision processes. 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. However, value function approximation is fundamentally different since target values are initially unknown and must be determined by solving the Bellman equation, for example by iteratively finding a fixed point of the Bellman backup operator. Furthermore, the set of samples of the manifold is not given a priori, but is determined through active exploration by the agent. Finally, in our work, basis functions can be constructed multiple times by interleaving policy improvement and representation learning. This is in spirit similar to the design of kernels adapted to regression or classification tasks (Szlam et al., 2006).

The rest of the paper is organized as follows. Section 2 gives a quick summary of the main framework called Representation Policy Iteration (RPI) for jointly learning represen-

5. The global nature of Fourier bases have been exploited in other areas, for example they have led to significantly improved algorithms for learning boolean functions (Jackson, 1995). Circuit designers have discovered fast algorithms for converting state-based truth-table and decision diagram representations of boolean functions into Fourier representations using the Hadamard transformation (Thornton et al., 2001). The eigenvectors of the graph Laplacian on boolean hypercubes form the columns of the Hadamard matrix (Bernasconi, 1998).

tations and policies, and illustrates a simplified version of the overall algorithm on the small two-room discrete MDP shown earlier in Figure 1. Section 3 motivates the use of the graph Laplacian from several different points of view, including as a spectral approximation of transition matrices, as well as inducing a smoothness regularization prior that respects the topology of the state space through a data-dependent kernel. Section 4 describes a concrete instance of the RPI framework using least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) as the underlying control learning method, and compares PVFs with two parametric bases—radial basis functions (RBFs) and polynomials—on small discrete MDPs. Section 5 describes one approach to scaling proto-value functions to large discrete product space MDPs, using the Kronecker sum matrix factorization method to decompose the eigenspace of the combinatorial Laplacian. This section also compares PVFs against RBFs on the *Blockers* task (Sallans and Hinton, 2004). Section 6 extends PVFs to continuous MDPs using the *Nyström* extension for interpolating eigenfunctions from sampled states to novel states. A detailed evaluation of PVFs in continuous MDPs is given in Section 7, including the inverted pendulum, the mountain car, and the Acrobot tasks (Sutton and Barto, 1998). Section 8 contains a brief description of previous work. Section 9 discusses several ongoing extensions of the proposed framework, including Kronecker product matrix factorization (Johns et al., 2007), multiscale diffusion wavelet bases (Mahadevan and Maggioni, 2006), and more elaborate diffusion models using directed graphs where actions are part of the representation (Johns and Mahadevan, 2007; Osentoski and Mahadevan, 2007).

2. Overview of The Framework

This section contains a brief summary of the overall framework, which we call *Representation Policy Iteration* (RPI) (Mahadevan, 2005b).[6] Figure 3 illustrates the overall framework. There are three main components: sample collection, basis construction, and policy learning. Sample collection requires a task specification, which comprises of a domain simulator (or alternatively a physically embodied agent like a robot), and an initial policy. In the simplest case, the initial policy can be a random walk, although it can also reflect a more informative hand-coded policy. The second phase involves constructing the bases from the collected samples using a diffusion model, such as an undirected (or directed) graph. This process involves finding the eigenvectors of a symmetrized graph operator such as the graph Laplacian. The final phase involves estimating the "best" policy representable in the span of the basis functions constructed (we are primarily restricting our attention to linear architectures, where the value function is a weighted linear combination of the bases). The entire process can then be iterated.

Figure 4 specifies a more detailed algorithmic view of the overall framework. In the sample collection 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 5 and Section 6. Given this set of samples, in the representation learning phase, an undirected (or directed) graph is constructed in one of several ways: two states can be

6. The term "Representation Policy Iteration" is used to succinctly denote a class of algorithms that jointly learn basis functions and policies. In this paper, we primarily use LSPI as the control learner, but in other work we have used control learners such as $Q(\lambda)$ (Osentoski and Mahadevan, 2007).

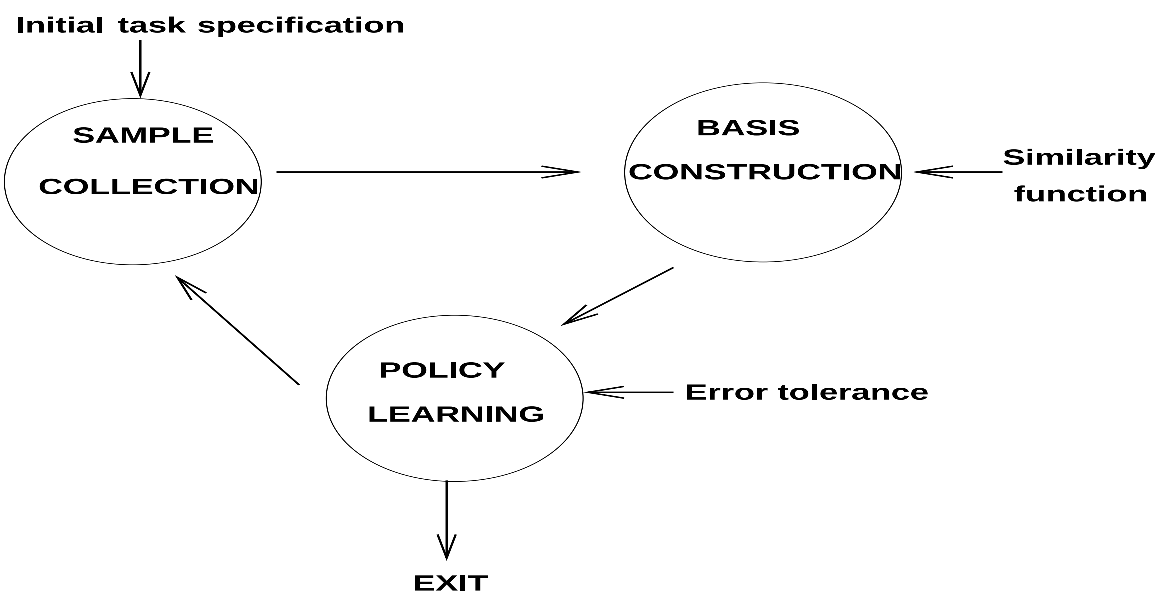


Figure 3: Flowchart of the unified approach to learning representation and behavior.

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 domains reported in Section 7. From the graph, proto-value functions are computed using one of the graph operators discussed below, for example the combinatorial or normalized Laplacian. 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 a model selection question, which will be empirically investigated in the experiments described later. 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 will be discussed in Section 9.

At the outset, it is important to point out that the algorithm described in Figure 4 is one of many possible designs that combine the learning of basis functions and policies. In particular, the RPI framework is an *iterative* approach, which interleaves the two phases of generating basis functions by sampling trajectories from policies, and then subsequently finding improved policies from the augmented set of basis functions. It may be possible to design alternative frameworks where basis functions are learned *jointly* with learning policies, by attempting to optimize some cumulative measure of optimality. We discuss this issue in more depth in Section 9.

`RPI` $(\pi_m, T, N, \epsilon, k, \mathcal{O}, \mu, \mathcal{D})$:

// π_m: Policy at the beginning of trial m
// T: Number of initial random walk trials
// N: Maximum length of each trial
// ϵ : Convergence condition for policy iteration
// k: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used
// μ: Parameter for basis adaptation
// $\mathcal{D}$: Initial set of samples

Sample Collection Phase

1. **Off-policy or on-policy sampling:** Collect a data set of samples $\mathcal{D}_m = \{(s_i, a_i, s_{i+1}, r_i), \ldots\}$ by either randomly choosing actions (off-policy) or using the supplied initial policy (on-policy) for a set of T trials, each of maximum N steps (terminating earlier if it results in an absorbing goal state), and add these transitions to the complete data set $\mathcal{D}$.

2. **(Optional) Subsampling step:** Form a subset of samples $\mathcal{D}_s \subseteq \mathcal{D}$ by some subsampling method such as random subsampling or trajectory subsampling. For episodic tasks, optionally prune the trajectories stored in $\mathcal{D}_s$ so that only those that reach the absorbing goal state are retained.

Representation Learning Phase

3. Build a diffusion model from the data in $\mathcal{D}_s$. In the simplest case of discrete MDPs, construct an undirected weighted graph G from $\mathcal{D}$ by connecting state i to state j if the pair (i, j) form temporally successive states $\in S$. Compute the operator $\mathcal{O}$ on graph G, for example the normalized Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.

4. Compute the k smoothest eigenvectors of $\mathcal{O}$ on the graph G. Collect them as columns of the basis function matrix Φ, a $|S| \times k$ matrix. The state action bases $\phi(s, a)$ can be generated from rows of this matrix by duplicating the state bases $\phi(s)$ $|A|$ times, and setting all the elements of this vector to 0 except for the ones corresponding to the chosen action.[a]

Control Learning Phase

5. Using a standard parameter estimation method (e.g., Q-learning or LSPI), find an ϵ-optimal policy π that maximizes the action value function $Q^\pi = \Phi w^\pi$ within the linear span of the bases Φ using the training data in $\mathcal{D}$.

6. **Optional:** Set the initial policy π_{m+1} to π and call `RPI` $(\pi_{m+1}, T, N, \epsilon, k, \mathcal{O}, \mu, \mathcal{D})$.

a. In large continuous and discrete MDPs, the basis matrix Φ need not be explicitly formed and the features $\phi(s, a)$ can be computed "on demand" as will be explained later.

Figure 4: This figure shows a generic algorithm for combining the learning of representation (or basis functions) from spectral analysis of random walks, and estimation of policies within their linear span. Elaborations of this framework will be studied in subsequent sections.

2.1 Sample Run of RPI on the Two-Room Environment

The result of running the algorithm is shown in Figure 5, which was obtained using the following specific parameter choices.

- The state space of the two room MDP is as shown in Figure 1. There are 100 states totally, of which 43 states are inaccessible since they represent interior and exterior walls. The remaining 57 states are divided into 1 doorway state and 56 interior room states. The agent is rewarded by +100 for reaching state 89, which is the last accessible state in the bottom right-hand corner of room 2, and its immediate neighbors. In the $3D$ value function plots shown in Figure 5, the axes are reversed to make it easier to visualize the value function plot, making state 89 appear in the top left (diagonally distant) corner.

- 3463 samples were collected using off-policy sampling from a random walk of 50 episodes, each of length 100 (or terminating early when the goal state was reached).[7] Four actions (compass direction movements) were possible from each state. Action were stochastic. If a movement was possible, it succeeded with probability 0.9. Otherwise, the agent remained in the same state. When the agent reaches state 89, it receives a reward of 100, and is randomly reset to an accessible interior state.

- An undirected graph was constructed from the sample transitions, where the weight matrix W is simply the adjacency $(0, 1)$ matrix. The graph operator used was the normalized Laplacian $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, where $L = D - W$ is referred to as the combinatorial Laplacian (these graph operators are described in more detail in Section 3).

- 20 eigenvectors corresponding to the smallest eigenvalues of $\mathcal{L}$ (duplicated 4 times, one set for each action) are chosen as the columns of the state action basis matrix Φ. For example, the first four eigenvectors are shown in Figure 5. These eigenvectors are *orthonormal*: they are normalized to be of length 1 and are mutually perpendicular. Note how the eigenvectors are sensitive to the geometric structure of the overall environment. For example, the second eigenvector allows partitioning the two rooms since it is negative for all states in the first room, and positive for states in the second room. The third eigenvector is non-constant over only one of the rooms. The connection between the Laplacian and regularities such as symmetries and bottlenecks is discussed in more detail in Section 3.6.

- The parameter estimation method used was least-squares policy iteration (LSPI), with $\gamma = 0.8$. LSPI is described in more detail in Section 4.1.

- The optimal value function using unit vector bases and the approximation produced by 20 PVFs are compared using the $2D$ array format in Figure 6.

7. Since the approximated value function shown in Figure 5 is the result of a specific set of random walk trajectories, the results can vary over different runs depending on the number of times the only rewarding (goal) state was reached. Section 4.2 contains more detailed experiments that measures the learned policy over multiple runs.

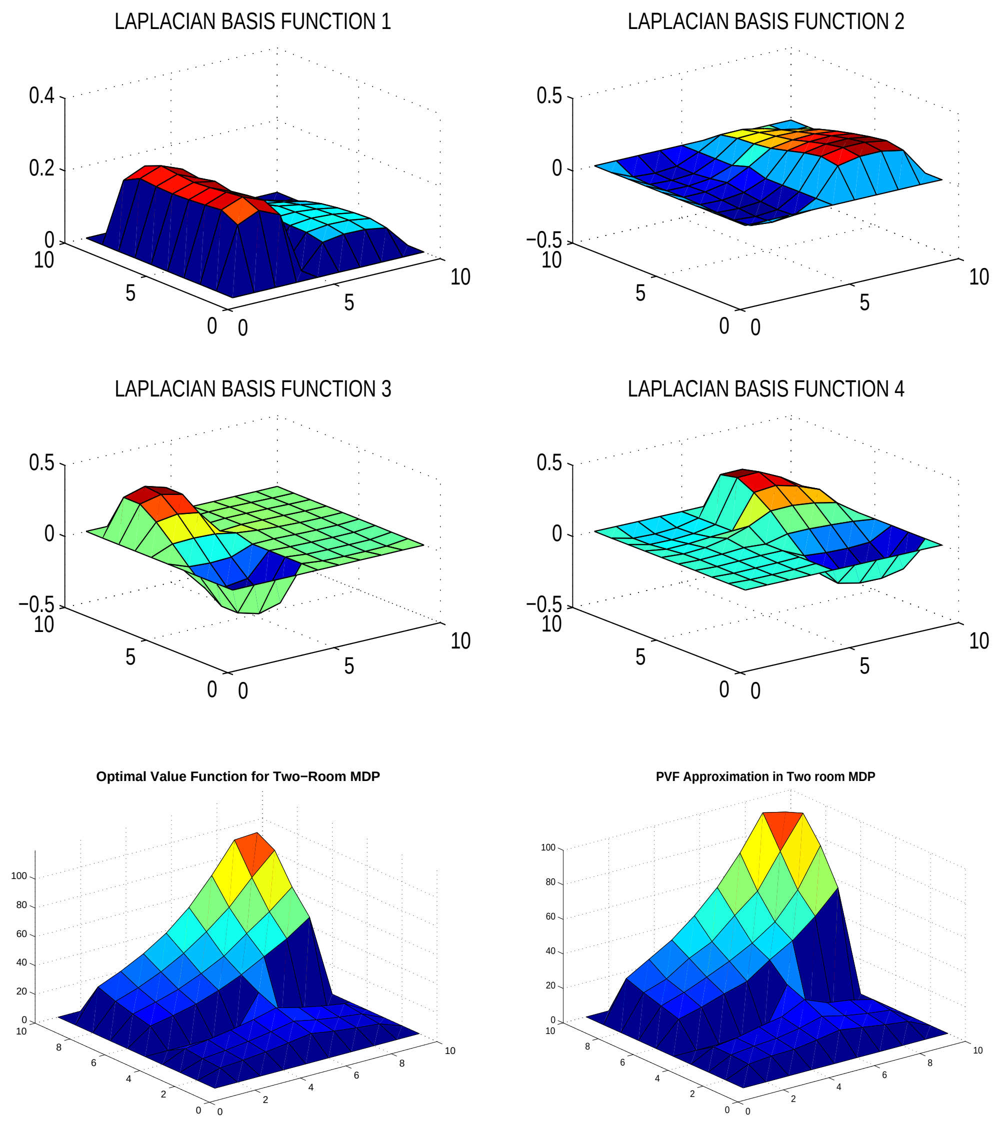


Figure 5: Top: proto-value functions formed from the four "smoothest" eigenvectors of the normalized graph Laplacian in a two-room MDP of 100 states. Bottom left: the optimal value function for a 2 room MDP, repeated from Figure 1 for comparative purposes. Bottom right: the approximation produced by the RPI algorithm using 20 proto-value functions, computed as the eigenvectors of the normalized graph Laplacian on the adjacency graph. The nonlinearity represented by the walls is clearly captured.

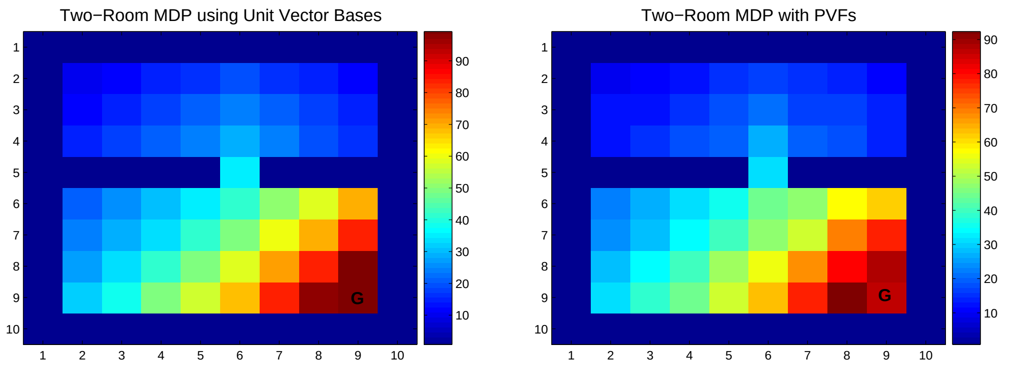


Figure 6: Left: the optimal value function for the two-room MDP using unit vector bases. Right: approximation with 20 PVFs using the RPI algorithm.

In the remainder of this paper, we will evaluate this framework in detail, providing some rationale for why the Laplacian bases are adept at approximating value functions, and demonstrating how to scale the approach to large discrete MDPs as well as continuous MDPs.

3. Representation Learning by Diffusion Analysis

In this section, we discuss the graph Laplacian, specifically motivating its use as a way to construct basis functions for MDPs. We begin with a brief introduction to MDPs, and then describe the spectral analysis of a restricted class of MDPs where the transition matrix is diagonalizable. Although this approach is difficult to implement for general MDPs, it provides some intuition into why eigenvectors are a useful way to approximate value functions. We then introduce the graph Laplacian as a symmetric matrix, which acts as a surrogate for the true transition matrix, but which is easily diagonalizable. It is possible to model non-symmetric actions and policies using more sophisticated symmetrization procedures (Chung, 2005), and we postpone discussion of this extension to Section 9. There are a number of other perspectives to view the graph Laplacian, namely as generating a data-dependent reproducing kernel Hilbert space (RKHS) (Scholkopf and Smola, 2001), as well as a way to generate nonlinear embeddings of graphs. Although a full discussion of these perspectives is beyond this paper, they are worth noting in order to gain deeper insight into the many remarkable properties of the Laplacian.

3.1 Brief Overview of MDPs

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 continuous Markov decision processes, the set of states $\subseteq \mathbb{R}^d$. Abstractly, a value function is a mapping $S \to \mathbb{R}$ or equivalently (in discrete MDPs) a vector $\in \mathbb{R}^{|S|}$. Given a policy $\pi : S \to A$ 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

$$V^*(s) = T^*(V^*(s)) = \max_a \left(R_{sa} + \gamma \sum_{s' \in S} P^a_{ss'} V^*(s') \right),$$

where $R_{sa} = \sum_{s' \in s} P^a_{ss'} R^a_{ss'}$ is the expected immediate reward. Value functions are mappings from the state space to the expected long-term discounted sum of rewards received by following a fixed (deterministic or stochastic) policy π. Here, T^* can be viewed as an *operator* on value functions, and V^* represents the fixed point of the operator T^*. The value function V^π associated with following a (deterministic) policy π can be defined as

$$V^\pi(s) = T(V^\pi(s)) = R_{s\pi(s)} + \gamma \sum_{s' \in S} P^{\pi(s)}_{ss'} V^\pi(s').$$

Once again, T is an operator on value functions, whose fixed point is given by V^π. Value functions in an MDP can be viewed as the result of rewards " diffusing" through the state space, governed by the underlying system dynamics. Let P^π represent an $|S| \times |S|$ transition matrix of a (deterministic) policy $\pi : S \rightarrow A$ mapping each state $s \in S$ to a desired action $a = \pi(s)$. Let R^π be a (column) vector of size $|S|$ of rewards. The value function associated with policy π can be defined using the Neumann series:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi = \left(I + \gamma P^\pi + \gamma^2 (P^\pi)^2 + \ldots\right) R^\pi. \quad (1)$$

3.2 Approximation of Value Functions

It is obviously difficult to represent value functions exactly on large discrete state spaces, or in continuous spaces. Consequently, there has been much study of approximation architectures for representing value functions (Bertsekas and Tsitsiklis, 1996). Value functions generally exhibit two key properties: they are typically *smooth*, and they reflect the underlying state space geometry. A fundamental contribution of this paper is the use of an approximation architecture that exploits a new notion of smoothness, not in the traditional sense of Euclidean space, but smoothness on the state space graph. The notion of smooth functions on graphs can be formalized using the Sobolev norm (Mahadevan and Maggioni, 2006). In addition, value functions usually reflect the geometry of the environment (as illustrated in Figure 5). Smoothness derives from the fact that the value at a given state $V^\pi(s)$ is always a function of values at "neighboring" states. Consequently, it is natural to construct basis functions for approximating value functions that share these two properties.[8]

Let us define a set of *basis functions* $F_\Phi = \{\phi_1, \ldots, \phi_k\}$, where each basis function represents a "feature" $\phi_i : S \rightarrow \mathbb{R}$. 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

8. For low values of the discount factor γ, it is possible to construct highly non-smooth value functions, which decay rapidly and are not influenced by nonlinearities in state space geometry. In many problems of interest, however, the discount factor γ needs to be set close to 1 to learn a desirable policy.

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,

$$V^\pi \approx \hat{V}^\pi = \Phi w^\pi = \sum_i w_i^\pi \phi_i \; .$$

Mathematically speaking, this problem can be rigorously formulated using the framework of best approximation in inner product spaces (Deutsch, 2001). In fact, it is easy to show that the space of value functions represents a Hilbert space, or a complete inner product space (Van Roy, 1998). For simplicity, we focus on the simpler problem of approximating a fixed policy π, which defines a Markov chain where ρ^π represents its invariant (long-term) distribution. This distribution defines a Hilbert space, where the inner product is given by

$$\langle V_1, V_2 \rangle_\pi = \sum_{s \in S} V_1^\pi(s) V_2^\pi(s) \rho^\pi(s).$$

The "length" or norm in this inner product space is defined as $\|V\|_\pi = \sqrt{\langle V, V \rangle_\pi}$. Value function approximation can thus be formalized as a problem of best approximation in a Hilbert space (Deutsch, 2001). It is well known that if the basis functions ϕ_i are *orthonormal* (unit-length and mutually perpendicular), the best approximation of the value function V^π can be expressed by its projection onto the space spanned by the basis functions, or more formally

$$V^\pi \approx \sum_{i \in I} \langle V^\pi, \phi_i \rangle_\pi \; \phi_i,$$

where I is the set of indices that define the basis set. In finite MDPs, the best approximation can be characterized using the weighted least-squares projection matrix

$$M_\Phi^\pi = \Phi(\Phi^T D_{\rho^\pi} \Phi)^{-1} \Phi^T D_{\rho^\pi},$$

where D_{ρ^π} is a diagonal matrix whose entries represent the distribution ρ^π. We know the Bellman "backup" operator T defined above has a fixed point $V^\pi = T(V^\pi)$. Many standard parameter estimation methods, including LSPI (Lagoudakis and Parr, 2003) and LSTD (Bradtke and Barto, 1996), can be viewed as finding an approximate fixed point of the operator T

$$\hat{V}^\pi = \Phi w^\pi = M_\phi^\pi \left(T(\Phi w^\pi) \right).$$

It can be shown that the operator T is a contraction mapping, where

$$\|TV_1 - TV_2\|_\pi \leq \gamma \|V_1 - V_2\|_\pi \; .$$

A natural question that arises is whether we can quantify the error in value function approximation under a set of basis functions F_ϕ. Exploiting the contraction property of the operator T under the norm defined by the weighted inner product, it can be shown that the "distance" between the true value function V^π and the fixed point $\hat{V}^\pi$ can be bounded in terms of the distance between V^π and its projection onto the space spanned by the basis functions (Van Roy, 1998):

$$\|V^\pi - \hat{V}^\pi\|_\pi \leq \frac{1}{\sqrt{1-\kappa^2}} \|V^\pi - M_\phi^\pi V^\pi\|_\pi \; ,$$

where κ is the contraction rate defined by Bellman operator T in conjunction with the weighted least-squares projection.

The problem of value function approximation in control learning is significantly more difficult, in that it involves finding an approximate fixed point of the initially unknown operator T^*. One standard algorithm for control learning is *approximate policy iteration* (Bertsekas and Tsitsiklis, 1996), which interleaves an *approximate policy evaluation* step of finding an approximation of the value function $\hat{V}^{\pi_k}$ associated with a given policy π_k at stage k, with a *policy improvement* step of finding the greedy policy associated with $\hat{V}^{\pi_k}$. Here, there are two sources of error introduced by approximating the exact value function, and approximating the policy. We will describe a specific type of approximate policy iteration method—the LSPI algorithm (Lagoudakis and Parr, 2003)—in Section 4, which uses a least-squares approach to approximate the action-value function. An additional problem in control learning is that the standard theoretical results for approximate policy iteration are often expressed in terms of the maximum (normed) error, whereas approximation methods are most naturally formulated as projections in a least-squared normed space. There continues to be work on developing more useful weighted least-square bounds, although these currently assume the policy is exactly representable (Munos, 2003, 2005). Also, it is possible to design approximation methods that directly carry out max-norm projections using linear programming, although this work usually assumes the transition dynamics is known (Guestrin et al., 2001),

Our approach to the problem of control learning involves finding a suitable set of basis functions by diagonalizing a learned diffusion model from sample trajectories, and to use projections in the Hilbert space defined by the diffusion model for policy evaluation and improvement. We first introduce the *Fourier* approach of finding basis functions by diagonalization, and then describe how diffusion models are used as a substitute for transition models. In Section 9, we will return to discuss other approaches (Petrik, 2007; Parr et al., 2007), where the Bellman operator T is used more directly in finding basis functions.

3.3 Spectral Analysis of Transition Matrices

In this paper, the orthogonal basis functions are constructed in the Fourier tradition by diagonalizing an operator (or matrix) and finding its *eigenvectors*. We motivate this approach by first assuming that the eigenvectors are constructed directly from a (known) state transition matrix P^π and show that if the reward function R^π is known, the eigenvectors can be selected nonlinearly based on expanding the value function V^π on the eigenvectors of the transition matrix. Petrik (2007) develops this line of reasoning, assuming that P^π and R^π are both known, and that P^π is diagonalizable. We describe this perspective in more detail below as it provides a useful motivation for why we use diagonalizable diffusion matrices instead. One subclass of diagonalizable transition matrices are those corresponding to *reversible* Markov chains (which will turn out to be useful below). Although transition matrices for general MDPs are *not* reversible, and their spectral analysis is more delicate, it will still be a useful starting point to understand diffusion matrices such as the graph Laplacian.[9] If the transition matrix P^π is diagonalizable, there is a complete set of eigenvectors $\Phi^\pi = (\phi^\pi_1, \dots \phi^\pi_n)$ that provides a change of basis in which the transition matrix P^π

9. In Section 9, we discuss extensions to more general non-reversible MDPs.

is representable as a diagonal matrix. For the sub-class of diagonalizable transition matrices represented by reversible Markov chains, the transition matrix is not only diagonalizable, but there is also an orthonormal basis. In other words, using a standard result from linear algebra, we have

$$P^\pi = \Phi^\pi \Lambda^\pi (\Phi^\pi)^T,$$

where Λ^π is a diagonal matrix of *eigenvalues*. Another way to express the above property is to write the transition matrix as a sum of *projection matrices* associated with each eigenvalue:

$$P^\pi = \sum_{i=1}^{n} \lambda_i^\pi \phi_i^\pi (\phi_i^\pi)^T,$$

where the eigenvectors ϕ_i^π form a complete orthogonal basis (i.e., $\| \phi_i^\pi \|_2 = 1$ and $\langle \phi_i^\pi, \phi_j^\pi \rangle = 0, i \neq j$). It readily follows that powers of P^π have the same eigenvectors, but the eigenvalues are raised to the corresponding power (i.e., $(P^\pi)^k \phi_i^\pi = (\lambda_i^\pi)^k \phi_i^\pi$). Since the basis matrix Φ spans all vectors on the state space S, we can express the reward vector R^π in terms of this basis as

$$R^\pi = \Phi^\pi \alpha^\pi, \tag{2}$$

where α^π is a vector of scalar weights. For high powers of the transition matrix, the projection matrices corresponding to the largest eigenvalues will dominate the expansion. Combining Equation 2 with the Neumann expansion in Equation 1, we get

$$\begin{aligned}
V^\pi &= \sum_{i=0}^{\infty} (\gamma P^\pi)^i \Phi^\pi \alpha^\pi \\
&= \sum_{i=0}^{\infty} \sum_{k=1}^{n} \gamma^i (P^\pi)^i \phi_k^\pi \alpha_k^\pi \\
&= \sum_{k=1}^{n} \sum_{i=0}^{\infty} \gamma^i (\lambda_k^\pi)^i \phi_k^\pi \alpha_k^\pi \\
&= \sum_{k=1}^{n} \frac{1}{1-\gamma \lambda_k^\pi} \phi_k^\pi \alpha_k^\pi \\
&= \sum_{k=1}^{n} \beta_k \phi_k^\pi,
\end{aligned}$$

where we used the property that $(P^\pi)^i \phi_j^\pi = (\lambda_j^\pi)^i \phi_j^\pi$. Essentially, the value function V^π is represented as a linear combination of eigenvectors of the transition matrix. In order to provide the most efficient approximation, we can truncate the summation by choosing some small number $m < n$ of the eigenvectors, preferably those for whom β_k is large. Of course, since the reward function is not known, it might be difficult to pick a priori those eigenvectors that result in the largest coefficients. A simpler strategy instead is to focus on those eigenvectors for whom the coefficients $\frac{1}{1-\gamma\lambda_k^\pi}$ are the largest. In other words, we should pick the eigenvectors corresponding to the *largest* eigenvalues of the transition matrix P^π (since the spectral radius is 1, the eigenvalues closest to 1 will dominate the smaller ones):

$$V^\pi \approx \sum_{k=1}^{m} \frac{1}{1-\gamma\lambda_k^\pi}\phi_k^\pi \alpha_k^\pi, \tag{3}$$

where we assume the eigenvalues are ordered in non-increasing order, so λ_1^π is the largest eigenvalue. If the transition matrix P^π and reward function R^π are both known, one can of course construct basis functions by diagonalizing P^π and choosing eigenvectors "out-of-order" (that is, pick eigenvectors with the largest β_k coefficients above). Petrik (2007) shows a (somewhat pathological) example where a linear spectral approach specified by Equation 3 does poorly when the reward vector is chosen such that it is orthogonal to the first k basis functions. It is an empirical question whether such pathological reward functions exhibit themselves in more natural situations. The repertoire of discrete and continuous MDPs we study seem highly amenable to the linear spectral decomposition approach. However, we discuss various approaches for augmenting PVFs with reward-sensitive bases in Section 9.

The spectral approach of diagonalizing the transition matrix is problematic for several reasons. One, the transition matrix P^π cannot be assumed to be symmetric, in which case one has to deal with complex eigenvalues (and eigenvectors). Second, we cannot assume that the transition matrix is known. Of course, one can always use samples of the underlying MDP generated by exploration to estimate the transition matrix, but the number of samples needed may be large. Finally, in control learning, the policy keeps changing, causing one to have to reestimate the transition matrix. What one would ideally like to have is a *surrogate* model that is easier to estimate than a full transition matrix, is always diagonalizable, and results in smooth basis functions that capture large scale geometry. *Diffusion models* serve to fulfill this role, as discuss next.

3.4 From Transition Matrices to Diffusion Models

We now develop a line of analysis where a graph is induced from the state space of an MDP, by sampling from a policy such as a random walk. Let us define a weighted graph $G = (V, E, W)$, where V is a finite set of vertices, and W is a weighted adjacency matrix with $W(i, j) > 0$ if $(i, j) \in E$, that is it is possible to reach state i from j (or vice-versa) in a single step. A simple example of a diffusion model on G is the random walk matrix $P_r = D^{-1}W$. Figure 7 illustrates a random walk diffusion model. Note the random walk matrix $P_r = D^{-1}W$ is not symmetric. However, it can be easily shown that P_r defines a *reversible* Markov chain, which induces a Hilbert space with respect to the inner product defined by the invariant distribution ρ:

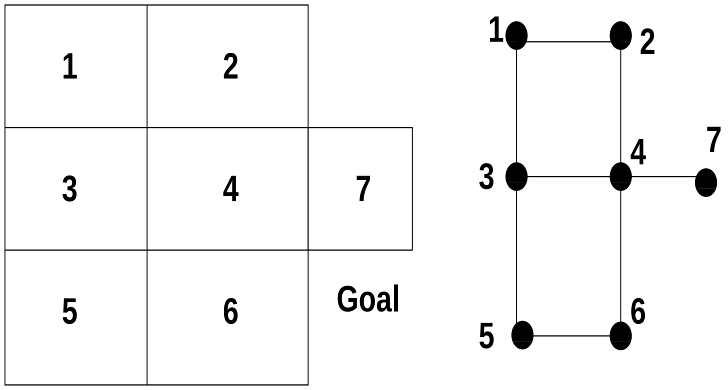
$$\langle f, g\rangle_\rho = \sum_{v\in V} f(i)g(i)\rho(i).$$

In addition, the matrix P_r can be shown to be self-adjoint (symmetric) with respect to the above inner product, that is

$$\langle P_r f, g\rangle_\rho = \langle f, P_r g\rangle_\rho.$$

Consequently, the matrix P_r can be shown to have real eigenvalues and orthonormal eigenvectors, with respect to the above inner product.

The random walk matrix $P_r = D^{-1}W$ is called a *diffusion model* because given any function f on the underlying graph G, the powers of $P_r^t f$ determine how quickly the random



$$P_r = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.33 & 0 & 0 & 0.33 & 0.33 & 0 & 0 \\ & & & \cdots & & & \end{bmatrix}$$

Figure 7: Top: A simple diffusion model given by an undirected unweighted graph connecting each state to neighbors that are reachable using a single (reversible) action. Bottom: first three rows of the random walk matrix $P_r = D^{-1}W$. P_r is not symmetric, but it has real eigenvalues and eigenvectors since it is spectrally related to the normalized graph Laplacian.

walk will "mix" and converge to the long term distribution (Chung, 1997). It can be shown that the stationary distribution of a random walk on an undirected graph is given by $\rho(v) = \frac{d_v}{vol(G)}$, where d_v is the degree of vertex v and the "volume" $vol(G) = \sum_{v \in G} d_v$. Even though the random walk matrix P_r can be diagonalized, for computational reasons, it turns out to be highly beneficial to find a symmetric matrix with a closely related spectral structure. This is essentially the graph Laplacian matrix, which we now describe in more detail.

3.5 The Graph Laplacian

For simplicity, assume the underlying state space is represented as an undirected graph $G = (V, E, W)$, where V is the set of vertices, E is the set of edges where $(u, v) \in E$ denotes an undirected edge from vertex u to vertex v. The more general case of directed graphs is discussed in Section 9.3. The *combinatorial Laplacian* L is defined as the operator $L = D - W$, where D is a diagonal matrix called the *valency* matrix whose entries are row sums of the weight matrix W. The first three rows of the combinatorial Laplacian matrix for the grid world MDP in Figure 7 is illustrated below, where we assume a unit weight on each edge:

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 \\ & & & \cdots & & & \end{bmatrix}.$$

Comparing the above matrix with the random walk matrix in Figure 7, it may seem like the two matrices have little in common. Surprisingly, we will show that there is indeed an intimate connection between the random walk matrix and the Laplacian. The Laplacian has many attractive spectral properties. It is both symmetric as well as positive semi-definite, and hence its eigenvalues are not only all real, but also non-negative. It is useful to view the Laplacian as an *operator* on the space of functions $\mathcal{F} : V \rightarrow \mathbb{R}$ on a graph. In particular, it can be easily shown that

$$Lf(i) = \sum_{j \sim i} (f(i) - f(j)),$$

that is the Laplacian acts as a *difference* operator. On a two-dimensional grid, the Laplacian can be shown to essentially be a discretization of the continuous Laplace operator

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

where the partial derivatives are replaced by finite differences.

Another fundamental property of the graph Laplacian is that projections of functions on the eigenspace of the Laplacian produce the smoothest global approximation respecting the underlying graph topology. More precisely, let us define the inner product of two functions f and g on a graph as $\langle f, g \rangle = \sum_u f(u)g(u)$.[10] Then, 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 (undirected) edges $u \sim v$ of the graph G, and w_{uv} denotes the weight on the edge. Note that each edge is counted only once in the sum. 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.

To make the connection between the random walk operator P_r introduced in the previous section, and the Laplacian, we need to introduce the *normalized Laplacian* (Chung, 1997), which is defined as

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

To see the connection between the normalized Laplacian and the random walk matrix $P_r = D^{-1}W$, note the following identities:

10. For simplicity, here we consider the unweighted inner product ignoring the invariant distribution ρ induced by a random walk.

$$\begin{aligned}
\mathcal{L} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}, \\
I - \mathcal{L} &= D^{-\frac{1}{2}} W D^{-\frac{1}{2}}, \\
D^{-\frac{1}{2}}(I - \mathcal{L}) D^{\frac{1}{2}} &= D^{-1} W.
\end{aligned}$$

Hence, the random walk operator $D^{-1}W$ is similar to $I - \mathcal{L}$, so both have the same eigenvalues, and the eigenvectors of the random walk operator are the eigenvectors of $I - \mathcal{L}$ point-wise multiplied by $D^{-\frac{1}{2}}$. We can now provide a rationale for choosing the eigenvectors of the Laplacian as basis functions. In particular, if λ_i is an eigenvalue of the random walk transition matrix P_r, then $1 - \lambda_i$ is the corresponding eigenvalue of $\mathcal{L}$. Consequently, in the expansion given by Equation 3, we would select the eigenvectors of the normalized graph Laplacian corresponding to the *smallest* eigenvalues.

The normalized Laplacian $\mathcal{L}$ also acts as a *difference* operator on a function f on a graph, that is

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

The difference between the combinatorial and normalized Laplacian is that the latter models the degree of a vertex as a local measure. In Section 7, we provide an experimental evaluation of the different graph operators for solving continuous MDPs.

Building on the Dirichlet sum above, a standard *variational* characterization of eigenvalues and eigenvectors views them as the solution to a sequence of minimization problems. In particular, the set of eigenvalues can be defined as the solution to a series of minimization problems using the Rayleigh quotient (Chung, 1997). This provides a variational characterization of eigenvalues using projections of an arbitrary function $g : V \to \mathcal{R}$ onto the subspace $\mathcal{L}g$. The quotient gives the eigenvalues and the functions satisfying orthonormality are the eigenfunctions:

$$\frac{\langle g, \mathcal{L} g \rangle}{\langle g, g \rangle} = \frac{\langle g, D^{-\frac{1}{2}} L D^{-\frac{1}{2}} g \rangle}{\langle g, g \rangle} = \frac{\sum_{u \sim v} (f(u) - f(v))^2 w_{uv}}{\sum_u f^2(u) d_u},$$

where $f \equiv D^{-\frac{1}{2}} g$. The first eigenvalue is $\lambda_0 = 0$, and is associated with the constant function $f(u) = \mathbf{1}$, which means the first eigenfunction $g_o(u) = \sqrt{D}\ \mathbf{1}$ (for an example of this eigenfunction, see top left plot in Figure 5). The first eigenfunction (associated with eigenvalue 0) of the combinatorial Laplacian is the constant function $\mathbf{1}$. The second eigenfunction is the infimum over all functions $g : V \to \mathcal{R}$ that are perpendicular to $g_o(u)$, which gives us a formula to compute the first non-zero eigenvalue λ_1, namely

$$\lambda_1 = \inf_{f \perp \sqrt{D}\mathbf{1}} \frac{\sum_{u \sim v} (f(u) - f(v))^2 w_{uv}}{\sum_u f^2(u) d_u}.$$

The Rayleigh quotient for higher-order basis functions is similar: each function is perpendicular to the subspace spanned by previous functions (see top four plots in Figure 5). In other words, the eigenvectors of the graph Laplacian provide a systematic organization of the space of functions on a graph that respects its topology.

3.6 Proto-Value Functions and Large-Scale Geometry

We now formalize the intuitive notion of why PVFs capture the large-scale *geometry* of a task environment, such as its symmetries and bottlenecks. A full discussion of this topic is beyond the scope of this paper, and we restrict our discussion here to one interesting property connected to the automorphisms of a graph. Given a graph $G = (V, E, W)$, an *automorphism* π of a graph is a bijection $\pi : V \rightarrow V$ that leaves the weight matrix invariant. In other words, $w(u, v) = w(\pi(u), \pi(v))$. An automorphism π can be also represented in matrix form by a permutation matrix Γ that commutes with the weight matrix:

$$\Gamma W = W \Gamma.$$

An immediate consequence of this property is that automorphisms leave the valency, or degree of a vertex, invariant, and consequently, the Laplacian is invariant under an automorphism. The set of all automorphisms forms a non-Abelian group, in that the group operation is non-commutative. Let x be an eigenvector of the combinatorial graph Laplacian L. Then, it is easy to show that Γx must be an eigenvector as well for any automorphism Γ. This result follows because

$$L \Gamma x = \Gamma L x = \Gamma \lambda x = \lambda \Gamma x.$$

A detailed graph-theoretic treatment of the connection between symmetries of a graph and its spectral properties are provided in books on algebraic and spectral graph theory (Chung, 1997; Cvetkovic et al., 1980, 1997). For example, it can be shown that if the permuted eigenvector Γx is independent of the original eigenvector x, then the corresponding eigenvalue λ has geometric multiplicity $m > 1$. More generally, it is possible to exploit the theory of linear representations of groups to construct compact basis functions on symmetric graphs, which have found applications in the study of complex molecules such as "buckyballs" (Chung and Sternberg, 1992). It is worth pointing out that these ideas extend to continuous manifolds as well. The use of the Laplacian in constructing representations that are *invariant* to group operations such as translation is a hallmark of work in harmonic analysis (Gurarie, 1992).

Furthermore, considerable work in spectral graph theory as well as its applications in AI uses the properties of the *Fiedler* eigenvector (the eigenvector associated with the smallest non-zero eigenvalue), such as its sensitivity to bottlenecks, in order to find clusters in data or segment images (Shi and Malik, 2000; Ng et al., 2002). To formally explain this, we briefly review spectral geometry. The *Cheeger* constant h_G of a graph G is defined as

$$h_G(S) = \min_S \frac{|E(S, \tilde{S})|}{\min(\text{vol } S, \text{vol } \tilde{S})}.$$

Here, S is a subset of vertices, $\tilde{S}$ is the complement of S, and $E(S, \tilde{S})$ denotes the set of all edges (u, v) such that $u \in S$ and $v \in \tilde{S}$. The volume of a subset S is defined as vol $S = \sum_{x \in S} d_X$. Consider the problem of finding a subset S of states such that the edge boundary ∂S contains as few edges as possible, where

$$\partial S = \{(u, v) \in E(G) : u \in S \text{ and } v \notin S\}.$$

The relation between ∂S and the Cheeger constant is given by

$$|\partial S| \geq h_G \ \text{ vol } S.$$

In the two-room grid world task illustrated in Figure 1, the Cheeger constant is minimized by setting S to be the states in the first room, since this will minimize the numerator $E(S, \tilde{S})$ and maximize the denominator $min(\text{vol } S, \text{vol } \tilde{S})$. A remarkable inequality connects the Cheeger constant with the spectrum of the graph Laplacian operator. This theorem underlies the reason why the eigenfunctions associated with the second eigenvalue λ_1 of the graph Laplacian captures the geometric structure of environments, as illustrated in Figure 5.

Theorem 1 *(Chung, 1997): Define λ_1 to be the first (non-zero) eigenvalue of the normalized graph Laplacian operator $\mathcal{L}$ on a graph G. Let h_G denote the Cheeger constant of G. Then, we have $2h_G \ \geq \ \lambda_1 > \frac{h_G^2}{2}$.*

In the context of MDPs, our work explores the construction of representations that similarly exploit large-scale geometric features, such as symmetries and bottlenecks. In other words, we are evaluating the hypothesis that such representations are useful in solving MDPs, given that topology-sensitive representations have proven to be useful across a wide variety of problems both in machine learning specifically as well as in science and engineering more generally.

4. Representation Policy Iteration

In this section, we begin the detailed algorithmic analysis of the application of proto-value functions to solve Markov decision processes. We will describe a specific instantiation of the RPI framework described previously, which comprises of an outer loop for learning basis functions and an inner loop for estimating the optimal policy representable within the given set of basis functions. In particular, we will use least-square policy iteration (LSPI) as the parameter estimation method. 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 5 and Section 6.

4.1 Least-Squares Approximation of Action Value Functions

The basics of Markov decision processes as well as value function approximation was briefly reviewed in Section 3. Here, we focus on action-value function approximation, and in particular, describe the LSPI method (Lagoudakis and Parr, 2003). In action-value learning, the goal is to approximate the true action-value function $Q^\pi(s, a)$ for a policy π using a set of basis functions $\phi(s, a)$ that can be viewed as doing dimensionality reduction on the space of functions. 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 $\in \mathbb{R}^{|S|\times|A|}$. $\phi(s,a)$ 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 tries to find a set of weights w^π under which the projection of the backed up approximated Q-function $T_\pi \hat{Q}^\pi$ onto the space spanned by the columns of Φ is a fixed point, namely

$$\hat{Q}^\pi = \Phi(\Phi^T\Phi)^{-1}\Phi^T(T_\pi \hat{Q}^\pi),$$

where T_π is the Bellman backup operator. It can be shown (Lagoudakis and Parr, 2003) that the resulting solution can be written in a *weighted* least-squares form as $Aw^\pi = b$, where the A matrix is given by

$$A = \left(\Phi^T D^\pi_\rho (\Phi - \gamma P^\pi \Phi)\right),$$

and the b column vector is given by

$$b = \Phi^T D^\pi_\rho R,$$

where D^π_ρ is a diagonal matrix whose entries reflect varying "costs" for making approximation errors on (s,a) pairs as a result of the nonuniform distribution $\rho^\pi(s,a)$ of visitation frequencies. A and b can be estimated from a database of transitions collected from some source, for example, a random walk. 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). Once the matrix A and vector b have been constructed, the system of equations $Aw^\pi = b$ can be solved for the weight vector w^π either by taking the inverse of A (if it is of full rank) or by taking its pseudo-inverse (if A is rank-deficient). This defines a specific policy since $\hat{Q}^\pi = \Phi w^\pi$. The process is then repeated, until convergence (which can be defined as when the $\mathbb{L}^2$- normed difference between two successive weight vectors falls below a predefined threshold ϵ). Note that in succeeding iterations, the A matrix will be different since the policy π has changed. Figure 8 describes a specific instantiation of RPI, using LSPI as the control learning method.

4.2 Evaluating RPI on Simple Discrete MDPs

In this section, we evaluate the effectiveness of PVFs using small discrete MDPs such as the two-room discrete MDP used above, before proceeding to investigate how to scale the

```
RPI ($\pi_m, T, N, \epsilon, k, \mathcal{O}, \mu, \mathcal{D}$):

// $\pi_m$: Policy at the beginning of trial $m$
// $T$: Number of initial random walk trials
// $N$: Maximum length of each trial
// $\epsilon$ : Convergence condition for policy iteration
// $k$: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used
// $\mu$: Parameter for basis adaptation

                         **Sample Collection Phase**

1. See Figure 4 on page 2177.

                       **Representation Learning Phase**

2. See Figure 4 on page 2177.

                      **Control Learning Phase (LSPI)**

3. Initialize $w^0 \in \mathbb{R}^k$ to a random vector.
4. **Repeat** the following steps:
    (a) Set $i \leftarrow i + 1$. Using the stored transitions $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}$, compute the matrix
        $A$ and vector $b$ as follows:
            $\tilde{A}^{t+1} = \tilde{A}^t + \phi(s_t, a_t)\left(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s_t))\right)^T.$
            $\tilde{b}^{t+1} = \tilde{b}^t + \phi(s_t, a_t) r_t.$
    (b) Solve the linear system of equations $\tilde{A}w^i = \tilde{b}$ using any standard method.[a]
    (c) **Optional basis adaptation step:** Prune the basis matrix $\Phi$ by discarding basis
        functions (columns) whose coefficients are smaller than $\mu$.
    (d) **until** $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
5. Set $\pi_{m+1}(s) = \text{argmax}_{a \in A} \hat{Q}^i(s, a)$ where $\hat{Q}^i = \Phi w^i$ is the $\epsilon$-optimal approximation to the
   optimal value function within the linear span of basis functions $\Phi$.
6. **Optional:** Repeat the above procedure by calling RPI ($\pi_{m+1}, T, N, \epsilon, k, \mathcal{O}, \mu, \mathcal{D}$).
```

a. If A is of full rank, it can be inverted, otherwise if it is rank-deficient, the pseudo-inverse of A can be used. It is possible to avoid matrix inversion entirely by using the incremental *Sherman-Woodbury-Morrison* method (Lagoudakis and Parr, 2003).

Figure 8: Pseudo-code of the representation policy iteration (RPI) using the least-squares policy iteration (LSPI) fix-point method as the control learning component.

framework to larger discrete and continuous MDPs.[11] PVFs are evaluated along a number of dimensions, including the number of bases used, and its relative performance compared to parametric bases such as polynomials and radial basis functions. In subsequent sections, we will probe the scalability of PVFs on larger more challenging MDPs.

Two-room MDP: The two-room discrete MDP used here is a 100 state MDP, where the agent is rewarded for reaching the top left-hand corner state in Room 2. As before, 57 states are reachable, and the remaining states are exterior or interior wall states. The state space of this MDP was shown earlier in Figure 1. Room 1 and Room 2 are both rectangular grids connected by a single door. There are four (compass direction) actions, each succeeding with probability 0.9, otherwise leaving the agent in the same state. The agent is rewarded by 100 for reaching the goal state (state 89), upon which the agent is randomly reset back to some starting (accessible) state.

Number of Basis Functions: Figure 9 evaluates the learned policy by measuring the number of steps to reach the goal, as a function of the number of training episodes, and as the number of basis functions is varied (ranging from 10 to 35 for each of the four actions). The results are averaged over 10 independent runs, where each run consisted of a set of training episodes of a maximum length of 100 steps, where each episode was terminated if the agent reached the absorbing goal state. Around 20 basis functions (per action) were sufficient to get close to optimal behavior, and increasing the number of bases to 35 produced a marginal improvement. The variance across runs is fairly small for 20 and 35 bases, but relatively large for smaller numbers of bases (not shown for clarity). Figure 9 also compares the performance of PVFs with unit vector bases (table lookup), showing that PVFs with 25 bases closely tracks the performance of unit vector bases on this task. Note that we are measuring performance in terms of the number of steps to reach the goal, averaged over a set of 10 runs. Other metrics could be plotted as well, such as the total discounted reward received, which may be more natural. However, our point is simply to show that there are significant differences in the quality of the policy learned by PVFs with that learned by the other parametric approximators, and these differences are of such an order that they will clearly manifest themselves regardless of the metric used.

Comparison with Parametric Bases: One important consideration in evaluating PVFs is how they compare with standard parametric bases, such as radial basis functions and polynomials. As Figure 1 suggests, parametric bases as conventionally formulated may have difficulty representing highly nonlinear value functions in MDPs such as the two room task. Here, we test whether this poor performance can be ameliorated by varying the number of basis functions used. Figure 9 evaluates the effectiveness of polynomial bases and radial basis functions in the two room MDP. In polynomial bases, a state i is mapped to the vector $\phi(i) = (1, i, i^2, \ldots i^{k-1})$ for k basis functions—this architecture was studied by (Koller and Parr, 2000; Lagoudakis and Parr, 2003).[12] In RBFs, a state i is mapped

11. In Section 9, we describe more sophisticated diffusion models for grid-world tasks in the richer setting of semi-Markov decision processes (SMDPs), using directed state-action graphs with temporally extended actions, such as "exiting a room", modeled with distal edges (Osentoski and Mahadevan, 2007).

12. The performance of polynomial bases gets worse for higher degrees, partly due to the numerical instability caused by taking large powers of state indices.

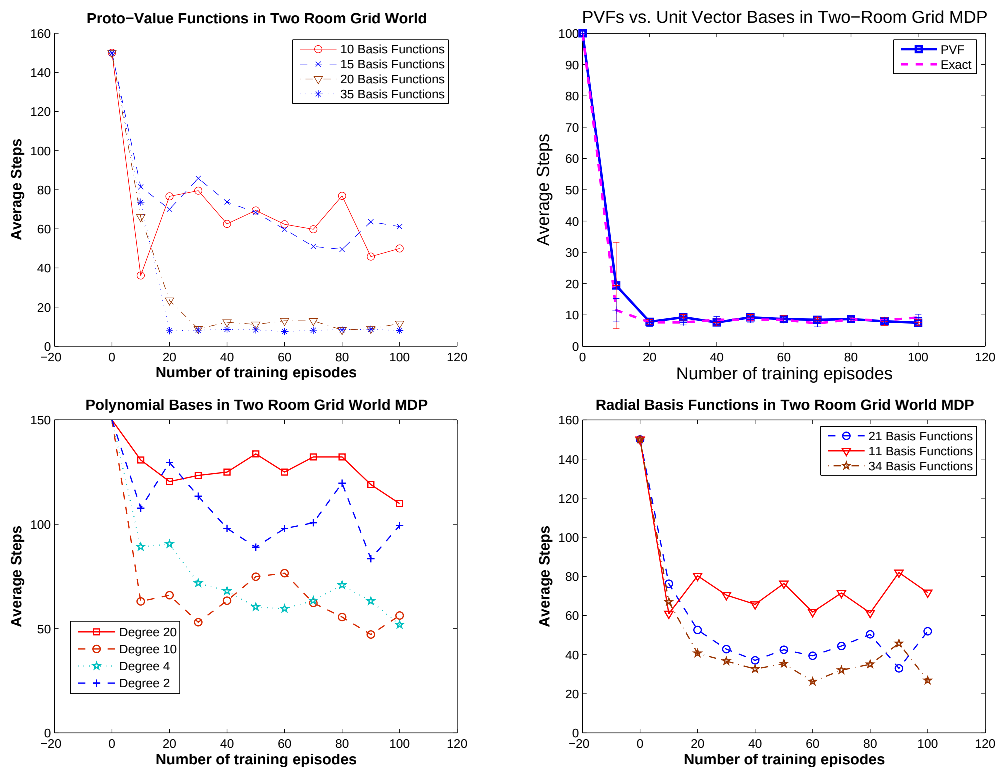


Figure 9: This experiment contrasts the performance of Laplacian PVFs (top left) with unit vector bases (top right), handcoded polynomial basis functions (bottom left) and radial basis functions (bottom right) on a 100 state two-room discrete MDP. Results are averaged over 10 runs. The performance of PVFs (with 25 bases) closely matches that of unit vector bases, and is considerably better than both polynomials and RBFs on this task.

to $\phi_j(i) = \exp^{-\frac{(i-j)^2}{2\sigma^2}}$, where j is the center of the RBF basis function. In the experiments shown, the basis centers were placed equidistantly from each other along the 100 states. The results show that both parametric bases under these conditions performed worse than PVFs in this task. [13]

Additional Results: Figure 10 shows an experiment on a larger 15×15 two-room MDP, with the same dynamics and goal structure as the smaller 10×10 two-room MDP. In this environment, there were a total of 225 states, with 157 of these being accessible interior states, and the remaining 68 representing "wall" states. Results are shown only for PVFs in this domain. The plotted result is averaged over 10 independent learning runs. As

13. Our results do not contradict any theoretical findings regarding the generality of RBFs or systems of orthogonal polynomials, since such results generally pertain to their asymptotic performance. Our evaluation of polynomials and RBFs gauges their performance on particular parameter settings.

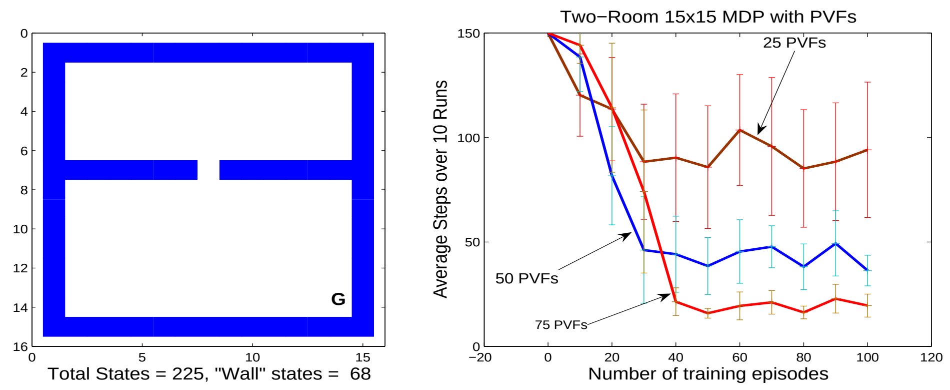


Figure 10: This figure shows results on a larger 15×15 two-room grid world MDP of 225 total states. The dynamics are identical to the two-room MDP. The results shown are using $25 - 75$ PVFs.

the number of PVFs is increased, the variance reduces and the performance significantly improves.

Figure 11 shows an additional experiment on a four-room MDP, where the agent is tasked to reach the state marked G. Results are shown only for PVFs in this domain. The plotted result is averaged over 10 independent learning runs. Here, the agent was trained on sample random walk trajectories that terminated in goal state G.

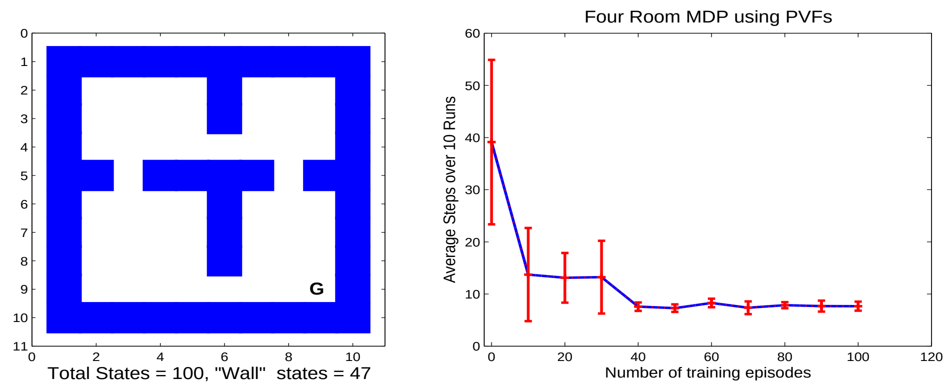


Figure 11: This figure shows results on a four-room grid world MDP of 100 total states. The dynamics are identical to the two-room MDP. The results shown are using 25 PVFs.

5. Scaling Proto-Value Functions: Product Spaces

Thus far, we have restricted our discussion of proto-value functions to small discrete MDPs. In this and the next section, we explore the issue of scaling the Laplacian framework to larger discrete and continuous domains. Computing and storing proto-value functions in large continuous or 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. To address this scaling issue, we explore a number of approaches, from exploiting the large-scale regular structure of product spaces described in this section, to the use of sparsification through sampling for continuous states described in the next section.

In this section, 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. A crucial property of the graph Laplacian is that its embeddings are highly regular for structured graphs (see Figure 13). We will explain the reason for this property below, and how to exploit it to construct compact encodings of Laplacian bases. We should also distinguish the approach described in this section, which relies on an *exact* Kronecker decomposition of the Laplacian eigenspace in product spaces, with the *approximate* Kronecker decomposition for arbitrary MDPs described in Section 9. The approach described here is applicable only to MDPs where the state space can be represented as the Kronecker sum of simpler state spaces (this notion will be defined more precisely below, but it covers many standard MDPs like grids). More generally, the weight matrices for arbitrary MDPs can also be factorized, although using the Kronecker *product*, where, however, the factorization is an approximation (Van Loan and Pitsianis, 1993).

5.1 Product Spaces: Complex Graphs from Simple Ones

Building on the theory of graph spectra (Cvetkovic et al., 1980), we now describe a hierarchical framework for efficiently computing and compactly storing proto-value functions. Many RL domains lead to *factored* representations where the state space is generated as the Cartesian product of the values of state variables (Boutilier et al., 1999). 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 on a large factored multiagent domain of $> 10^6$ states, where proto-value functions are constructed from diagonalizing Laplacian matrices of size only 100×100, a huge computational savings! Figure 12 illustrates the idea of scaling proto-value functions to large product spaces.[14]

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 of graphs. Let $G_1, \dots, G_n$ be n undirected graphs whose corresponding vertex and edge sets are specified as $G_i = (V_i, E_i)$. The *Kronecker sum graph*

14. Even greater reduction in the size of PVFs can be realized by exploiting the group invariance property of Laplacian operators, as described in Section 3.6. In particular, the graphs shown in Figure 12 have large automorphism groups, which can be exploited in significantly reducing the size of the corresponding Laplacian eigenspaces.

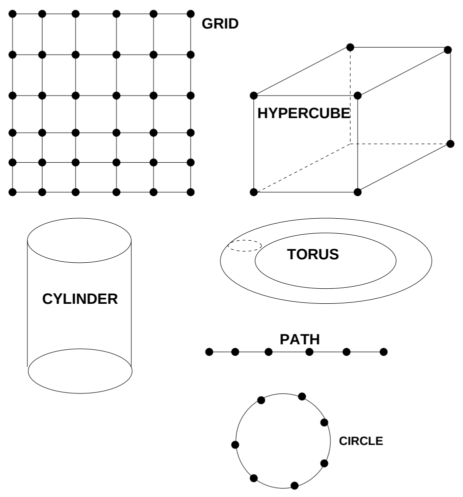


Figure 12: The spectrum and eigenspace of structured state spaces, including grids, hypercubes, cylinders, and tori, can be efficiently computed from "building block" subgraphs, such as paths and circles. Applied to MDPs, this hierarchical framework greatly reduces the computational expense of computing and storing proto-value functions.

$G = G_1 \oplus \ldots \oplus G_n$ 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 12 is the *Kronecker sum* of two path graphs; the hypercube is the Kronecker sum of three or more path graphs.

The Kronecker sum graph can also be defined using operations on the component adjacency matrices. If A_1 is a (p, q) matrix and A_2 is a (r, s) matrix, the Kronecker product matrix [15] $A = A_1 \otimes A_2$ 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. 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. The main result that we will exploit 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. The following result is well-known in the literature (Graham, 1981).

Theorem 2 *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*

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

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

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

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

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. We denote the set of eigenvectors of an operator $\mathcal{T}$ by the notation $X(\mathcal{T})$ and its spectrum by $\Sigma(\mathcal{T})$. A standard result that follows from the above theorem shows that the combinatorial graph Laplacian of a Kronecker sum of two graphs can be computed from the Laplacian of each subgraph.[16]

Theorem 3 *If $L_1 = L(G_1)$ and $L_2 = L(G_2)$ are the combinatorial Laplacians of graphs $G_1 = (V_1, E_1, W_1)$ and $G_2 = (V_2, E_2, W_2)$, 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*

$$(\Sigma(L), X(L)) = \{\lambda_i + \kappa_j, l_i \otimes k_j\},\ 1 \leq i \leq |V_1|, 1 \leq j \leq |V_2|,$$

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. Figure 13 illustrates this theorem, showing that the eigenvectors of the combinatorial Laplacian produce a regular embedding of a grid in 2D as well as a cylinder in 3D. These figures were generated as follows. For the grid shown on the left, the eigenvectors were generated as the Kronecker product of the eigenvectors of the combinatorial Laplacian for two chains of size 10. The figure shows the embedding of the grid graph where each state was embedded in $\mathbb{R}^2$ using the second and third smallest eigenvector. For the cylinder on the right, the eigenvectors were generated as the Kronecker product of the eigenvectors of the combinatorial Laplacian for a 10 state closed chain and a 5 state open chain. The embedding of the cylinder shown on the right was produced using the third and fourth eigenvector of the combinatorial Laplacian.

For the combinatorial Laplacian, 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 graph as well. Furthermore, for each eigenvector v_i, the Kronecker product $v_i \otimes \mathbf{1}$ will also be an eigenvector of the sum graph. One consequence of these properties is that geometry is well preserved, so for example the combinatorial Laplacian produces well-defined embeddings of structured spaces. Figure 13 shows the embedding of a cylinder (Kronecker sum of a closed and open chain) under the combinatorial Laplacian.

16. In contrast, the normalized Laplacian is not well-defined under sum, but has a well-defined semantics for the Kronecker or direct product of two graphs. The Kronecker product can also be used as a general method to approximate any matrix by factorizing it into the product of smaller matrices. We discuss the use of this approach to scaling PVFs in Section 9.

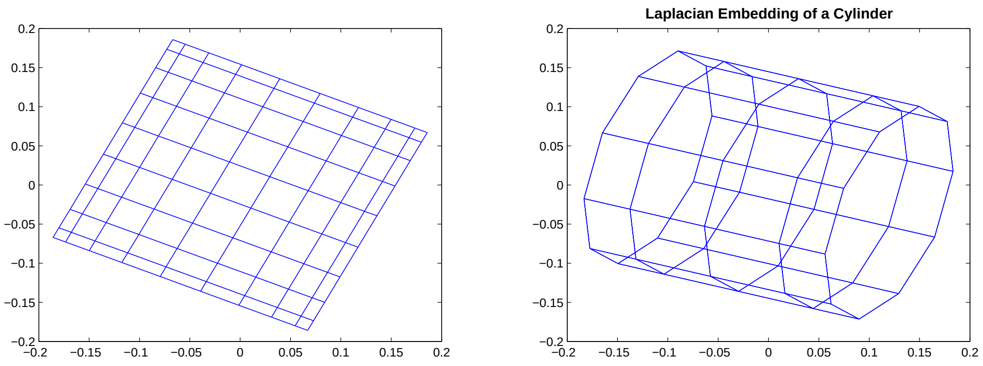


Figure 13: Left: this figure shows an embedding in $\mathbb{R}^2$ of a 10×10 grid world environment using "low-frequency" (smoothest) eigenvectors of the combinatorial Laplacian, specifically those corresponding to the second and third smallest eigenvalues. Right: the embedding of a "cylinder" graph using two low-order eigenvectors (3^{rd} and 4^{th}) of the combinatorial Laplacian. The cylinder graph is the Kronecker sum of a closed and open chain graph.

5.2 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 graph 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 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:

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

The corresponding update equation for the reward component is:

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

5.3 Experimental Results

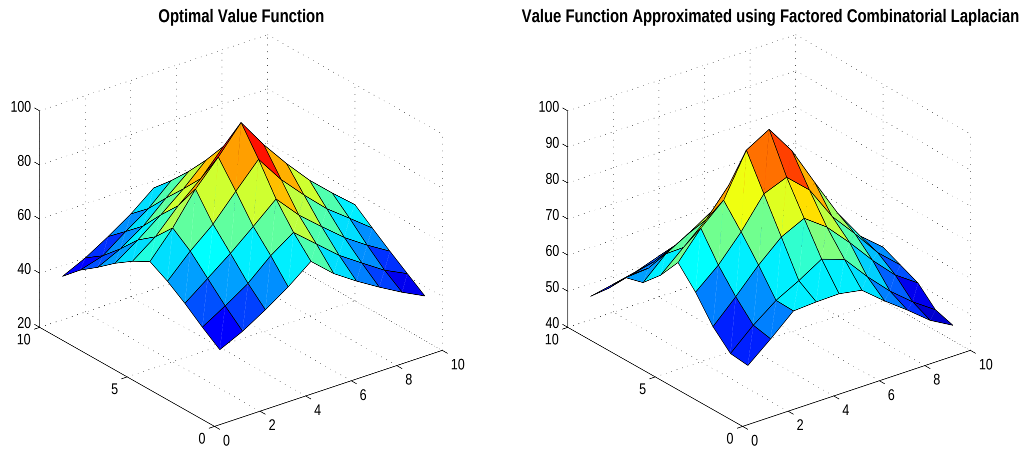


Figure 14: Left: the exact value function on a 10×10 grid world with a reward of +100 at the center. Right: a factored (combinatorial) Laplacian approximation using basis functions constructed by taking Kronecker products of basis functions for chain graphs (of length corresponding to row and column sizes).

To illustrate the Kronecker factorization presented in the previous section, we begin with a simple MDP. Figure 14 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 PVFs are computed as the Kronecker product of the PVFs on a 10 node chain graph, which requires both significantly smaller space of size $10 \times k$ for k basis functions, and much less computational effort (diagonalizing a Laplacian of size

10×10). These computational savings obviously magnify in larger grid world domains. In a grid world with 10^6 states, "flat" proto-value functions require $k \times 10^6$ space and time proportional to $(10^6)^3$ to be computed, whereas the factored basis functions only require space $k \times 10^3$ to store with much less computational cost to find.

5.4 The Blocker Task

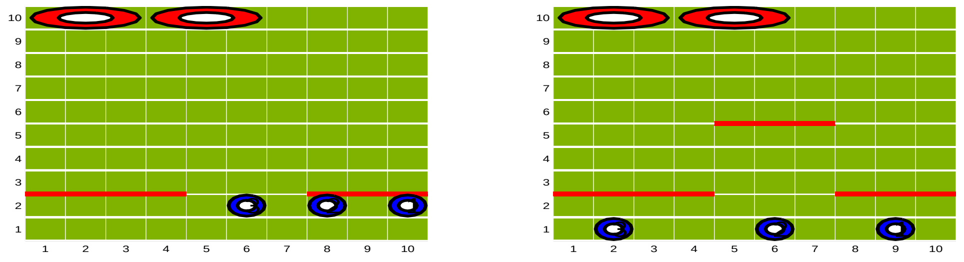


Figure 15: Two versions of the blocker domain are shown, each generating a state space of $> 10^6$ states. Interior walls shown create an "irregular" factored MDP whose overall topology can be viewed as a "perturbed" variant of a pure product of grids or cylinders (for the "wrap-around" case).

We now present a detailed study using a much larger factored multiagent domain called the "Blockers" task, which was first proposed by Sallans and Hinton (2004). This task, illustrated in Figure 15, 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 Kronecker product bases to "irregular" grids whose topology deviates from a pure hypercube or toroid. In the first variant, shown on the left in Figure 15, 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.[17]

The basis functions for the overall Blocker state space were computed as Kronecker products of the basis functions over each agent's state space. Each agent's state space was modeled as a grid (as in Figure 14) or a cylinder (for the "wrap-around" case). Since the presence of interior walls obviously violates the pure product of cylinders or grids topology, each individual agent's state space was learned from a random walk. The overall basis

17. In the Blocker domain, the interior walls are modeled as having "zero width", and hence all 100 states in each grid remain accessible, unlike the two-room environment.

functions were then constructed as Kronecker products of Laplacian basis functions for each learned (irregular) state grid.

Figure 16 compares the performance of the factored Laplacian bases with a set of radial basis functions (RBFs) for the first Blocker domain (shown on the left in Figure 15). 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 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 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 faster, but learn a worse policy. The factored Laplacian bases converge slower than RBFs, but learn a substantially better policy. Figure 16 also shows results for the second Blocker domain (shown on the right in Figure 15 with both side and interior middle walls), comparing 100 factored Laplacian bases with a similar number of RBFs. The results show a significant improvement in performance of the factored Laplacian bases over RBFs.

In terms of both space and time, the factored approach greatly reduces the computational complexity of finding and storing the Laplacian bases. A worst-case estimate of the size of the full Laplacian matrix is $O(|S|^2)$. Diagonalizing a $|S| \times |S|$ symmetric matrix and finding k eigenvectors requires time $O(k|S|^2)$ and $O(k|S|)$ space. Instantiating these general estimates for the Blocker domain, let n refer to the number of rows and columns in each agent's state space ($n = 10$ in our experiments), and k refer to the number of basis functions ($k = 100$ in our experiments). Then, the size of the state space is $|S| = (n^2)^3$, implying that the non-factored approach requires $O(k(n^2)^3)$ space and $O(k(n^6)^2)$ time, whereas the factored approach requires $O(kn^2)$ space and $O(k(n^2)^2)$ time. Note these are worse-case estimates. The Laplacian matrix is in fact highly sparse in the Blocker domain, requiring far less than $O(|S|^2)$ space to be stored. In fact, even in such a deterministic MDP where the Laplacian matrix can be stored in $O(|S|)$ space, the non-factored approach will still take $O(kn^3)$ space and $O(kn^6)$ time, whereas the factored approach takes $O(kn)$ space and $O(kn^2)$ time.

6. Scaling Proto-Value Functions: 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).

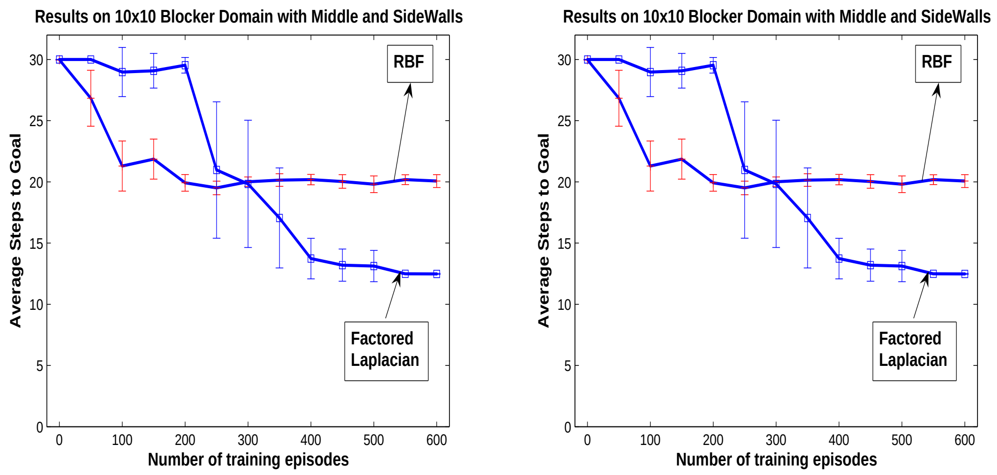


Figure 16: Comparison of factored (Laplacian) PVF basis functions with hand coded radial basis functions (RBF) on a 10×10 "wrap-around" grid with 3 agents and 2 blockers of $> 10^6$ states. Both approaches were tested using 100 basis functions. The plots show performance of PVFs against RBFs on the two blocker domains in Figure 15.

There is a rich and well-developed theory of the Laplace operator on manifolds, which we can only briefly summarize here. The Laplace-Beltrami operator has been extensively studied in the general setting of Riemannian manifolds (Rosenberg, 1997). Riemannian manifolds have been actively studied recently in machine learning in several contexts, namely in the context of designing 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).

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, 2006), unlike here where only samples of a manifold are given.

6.1 Nyström Extension

To learn policies on continuous MDPs, it is necessary to be able to extend eigenfunctions computed on a set of points $\in \mathbb{R}^n$ 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). The eigenfunction problem can be stated as

$$\int_D K(x,y)\phi(y)dy = \lambda\phi(x), \forall x \in D, \tag{4}$$

where D can be any domain, for example, $\mathbb{R}$. Using the standard quadrature approximation, the above integral can be written as

$$\int_D K(x,y)\phi(y)dy \approx \sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}(s_i), \tag{5}$$

where w_i are the quadrature weights, s_i are n selected sample points, and $\hat{\phi}$ is an approximation to the true eigenfunction. Combining Equation 4 and Equation 5 gives us

$$\sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}(s_i) = \hat{\lambda}\hat{\phi}(x).$$

By letting x denote any set of n points, for example the set of quadrature points s_i itself, the kernel $k(s_i,s_j)$ becomes a symmetric matrix. This enables computing the approximate eigenfunction at any new point as

$$\hat{\phi}_m(x) = \frac{1}{\hat{\lambda}} \sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}_m(s_i). \tag{6}$$

Let us instantiate Equation 6 in the context of the normalized Laplacian $\mathcal{L} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$. First, note that if λ_i is an eigenvalue of $\mathcal{L}$, then $1-\lambda_i$ is the corresponding eigenvalue of the diffusion matrix $D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$. Applying the the Nyström extension for computing the eigenfunctions of the normalized Laplacian $\mathcal{L}\phi_i = \lambda_i\phi_i$, we get the equation

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

where $d(z) = \sum_{y\sim z} w(z,y)$, and x is a new vertex in the graph. Note that the weights $w(x,y)$ from the new state x to its nearest neighbors y in the previously stored samples is determined at "run time" using the same nearest neighbor weighting algorithm used to compute the original weight matrix W. An extensive discussion of the Nyström method is given in Drineas and Mahoney (2005), and more details of its application to learning control in MDPs are given in Mahadevan et al. (2006).

Figure 17 illustrates the basic idea. Note that the Nyström method does *not* require recalculating eigenvectors—in essence, the embedding of a new state is computed by averaging over the already computed embeddings of "nearby" states. In practice, significant speedups can be exploited by using the following optimizations. We have empirically observed that roughly only 10% of the overall samples needed for learning a good policy are necessary to construct basis functions. Once the bases is defined over these sub-sampled states, the Nyström extended embeddings of the remaining 90% of training samples needs to be calculated only once, and henceforth can be cached during repeated runs of policy iteration. During testing, the Nyström embeddings of novel states encountered must be computed, but since the eigenvectors are defined over a relatively small core set of sample states, the extensions can be computed very efficiently using a fast nearest neighbor algorithm.[18]

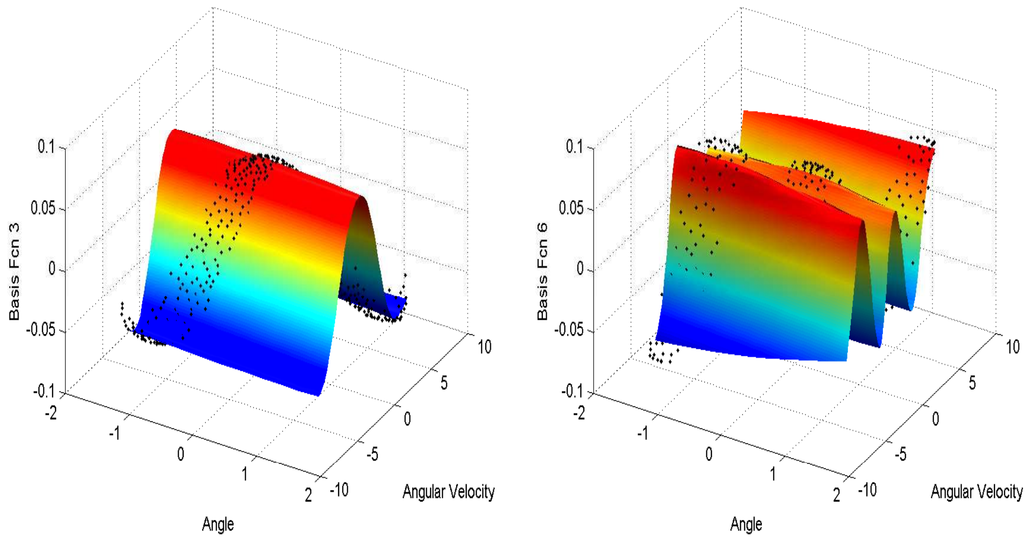


Figure 17: This figure illustrates the Nyström interpolation method for extending eigenfunctions on samples to new states. Left: the 3rd eigenvector of the Laplacian plotted on a set of samples (shown as filled dots) drawn from a random walk in the inverted pendulum domain, as well as its Nyström interpolated values. Right: the Nyström interpolated 6th eigenvector illustrated the entire state space as well as on the actual samples (again shown as filled dots).

6.2 Representation Policy Iteration for Continuous Domains

Figure 18 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 7 reveal. In Figure 18, $\mathcal{D}_{\mathcal{Z}}$ denotes the subsampled set of states. 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 18), as well as new states encountered during the testing of a learned policy.

6.3 Sampling from Point Sets $\in \mathbb{R}^n$

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 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

18. In our experiments, we used the TSTOOLS MATLAB nearest neighbor package.

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

// $\pi_m$: Initial policy
// $T$: Number of initial random walk trials
// $N$: Maximum length of each trial
// $\epsilon$ : Convergence condition for policy iteration
// $k$: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used
// $\mathcal{D}$: Data set of transitions

                    Sample Collection Phase

  1. See Figure 4 on page 2177.

                    Representation Learning Phase

  2. Build an undirected weighted graph $G$ from the set of subsampled transitions $\mathcal{D}_s \subseteq \mathcal{D}$ using
     the method described in Section 6.4 on graph construction from point sets $\in \mathbb{R}^n$. Compute
     the operator $\mathcal{O}$ on graph $G$ as discussed in Section 6.4.
  3. Compute the $k$ "smoothest" eigenvectors of $\mathcal{O}$ on the sub-sampled graph $\mathcal{D}_s$, and collect
     them as columns of the basis function matrix $\Phi$, a $|\mathcal{D}_s| \times k$ matrix. The embedding of a
     state action pair $\phi(s,a)$ where $s \in \mathcal{D}_s$ 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 Kronecker product.

                    Control Learning Phase:

  4. See Figure 8 on page 2192. For all transitions involving a state $s \notin \mathcal{D}_s$, its embedding is
     computed using the Nyström extension described in Section 6.1.
  5. Optional: Repeat the above procedure by calling RPI $(\pi_{m+1}, T, N, \epsilon, k, \mathcal{O}, \mu, \mathcal{D})$.
```

Figure 18: Pseudo-code of the representation policy iteration algorithm for continuous MDPs.

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 19 illustrates two methods for subsampling in the mountain car domain, including random subsampling and trajectory-based subsampling. The trajectory-based algorithm follows a greedy strategy: starting with the null set, add samples to the subset that are not within a specified distance to any sample currently in the subset. A maximal subset is returned when no more samples can be added. 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.

More formally, the trajectory based subsampling algorithm works as follows. We define an ϵ-net of points in $\mathcal{S}'$ to be a subset $\mathcal{S}''$ such that no two points are closer than ϵ, and that for every point y in $\mathcal{S}'$, there is a point in $\mathcal{S}''$ which is not farther than ϵ from y. One

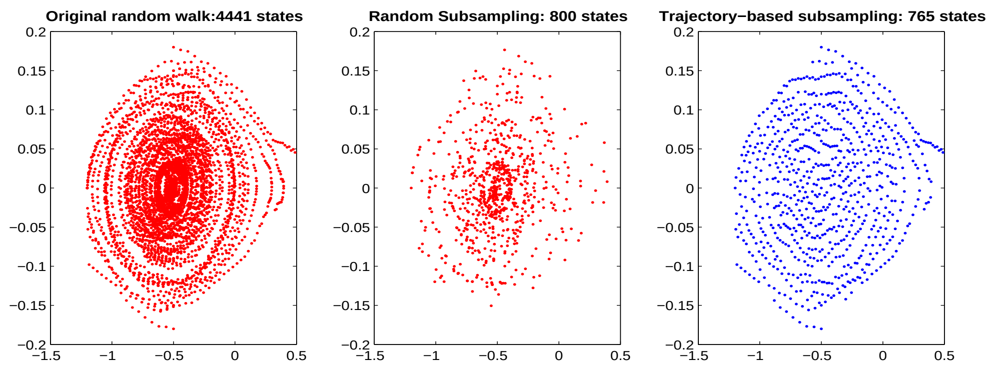


Figure 19: The problem of subsampling is illustrated in the mountain car domain. On the left is shown the original states visited during a random walk. In the middle is the subsampled data using a random subsampling algorithm. On the right is a trajectory based subsampling method.

can construct a (random) ϵ-net in $\mathcal{S}'$ as follows. Pick $x_0 \in \mathcal{S}'$ at random. By induction, for $k \geq 1$ suppose $x_0, x_1, \ldots, x_k$ have been picked so that the distance between any pair is larger than ϵ. If

$$R_k := \mathcal{S}' \setminus (\cup_{l=1}^{k} B_\epsilon(x_l))$$

is empty, stop, otherwise pick a point x_{k+1} in R_k. By definition of R_k the distance between x_{k+1} and any of the points $x_0, \ldots, x_k$ is not smaller than ϵ. When this process stops, say after k^* points have been selected, for any $y \in \mathcal{S}'$ we can find a point in $\mathcal{S}''$ not farther than ϵ, for otherwise $y \in R_{k^*}$ and the process would not have stopped.

In the experiments reported in Section 7, where states are continuous vectors $\in \mathbb{R}^n$, typically $< 10\%$ of the transitions in the original set of random walks are necessary to learn an adequate set of basis functions. For example, in the mountain car task, around 700 samples are sufficient to form the basis functions, whereas usually > 7000 samples are needed to learn a close to optimal policy.[19]

6.4 Graph Construction from Point Sets $\in \mathbb{R}^n$

Given a data set $\{x_i\}$ in $\mathbb{R}^n$, 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. In the experiments below, the following construction was used. Edges were inserted between a pair of states x_i and x_j if:

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

Weights were assigned to the edges in the following way:

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

19. In Section 9, we describe how Kronecker factorization can be used to significantly compress the size of the basis matrices.

Observe that for undirected graphs, 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, the above construction will still yield asymmetric weight matrices. We used an additional symmetrization step where we replaced the weight matrix W constructed by the symmetric $W + W^T$. If the states $\{x_i\}$ are drawn uniformly from a Riemannian manifold, then it is shown in Belkin and Niyogi (2004) that the above construction, 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 in MDPs 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 the above construction yields an approximation to the Laplace-Beltrami operator. Various ways of normalizing the weight matrix were explored in our experiments in Section 7. In particular, we compared the normalized Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ and the combinatorial Laplacian, $L = D - W$ operators.

7. Fully Interleaved Representation and Policy Learning: Continuous MDPs

In this section, we present a detailed analysis of fully interleaved representation and policy learning on continuous MDPs. By "fully interleaved", we mean that the overall learning run is divided into a set of discrete episodes of sample collection, basis construction, and policy learning. At the end of each episode, a set of additional samples is collected using either a random walk (off-policy) or the currently best performing policy (on-policy), and then basis functions are then recomputed and a new policy is learned. In all the experiments below, the trajectory based method was used to build the graph from which proto-value functions were learned. We discuss alternate approaches for interleaving basis function generation and control learning in Section 9.

7.1 Three Control Tasks

We explored the effectiveness and stability of proto-value functions in three continuous domains—the Acrobot task, the inverted pendulum task, and the mountain car task—that have long been viewed as benchmarks in the field. These three domains are now described in more detail.

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.

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

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.

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

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

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

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.

The Acrobot Task: The Acrobot task (Sutton and Barto, 1998) is a two-link under-actuated robot that is an idealized model of a gymnast swinging on a highbar. The only action available is a torque on the second joint, discretized to one of three values (positive, negative, and none). The reward is -1 for all transitions leading up to the goal state. The detailed equations of motion are given in Sutton and Barto (1998). The state space for the Acrobot is 4-dimensional. Each state is a 4-tuple represented by $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$. θ_1 and θ_2 represent the angle of the first and second links to the vertical, respectively, and are naturally in the range $(0, 2\pi)$. $\dot{\theta}_1$ and $\dot{\theta}_2$ represent the angular velocities of the two links. Notice that angles near 0 are actually very close to angles near 2π due to the rotational symmetry in the state space.

Figure 20 plots the Acrobot state space projected onto the subspace spanned by the two joint angles θ_1 and θ_2. This subspace is actually a torus. To approximate computing distances on the torus, the original states were projected upwards to a higher dimensional state space $\subset \mathbb{R}^6$ by mapping each angle θ_i to $(\sin(\theta_i), \cos(\theta_i))$. Thus, the overall state space is now $(\sin(\theta_1), \cos(\theta_1), \dot{\theta}_1, \sin(\theta_2),$
$\cos(\theta_2), \dot{\theta}_2)$. The motivation for this remapping is that now Euclidean distances in this augmented space better approximate local distances on the torus. In fact, ignoring the wrap-around nature of the Acrobot state space by simply using a local Euclidean distance metric on the four-dimensional state space results in significantly poorer performance. This example illustrates how overall global knowledge of the state space, just like in the Blockers domain, is valuable in designing a better local distance function for learning PVFs. This

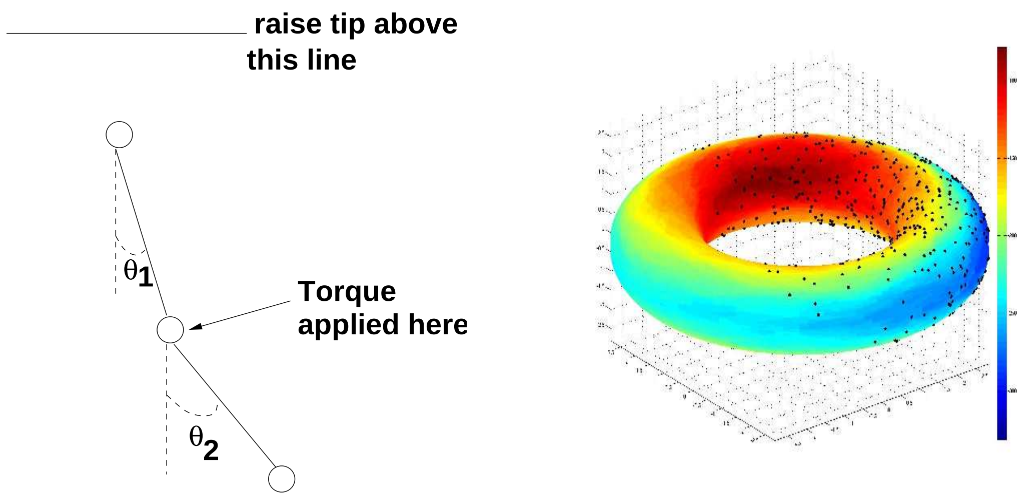


Figure 20: The state space of the Acrobot (shown on the left) exhibits rotational symmetries. The figure on the right plots its projection onto the subspace of $\mathbb{R}^2$ spanned by the two joint angles θ_1 and θ_2, which can be visualized as a torus. The angular velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ were set to 0 for this plot. The points shown on the torus are subsampled states from a random walk. The colors indicate the value function, with red (darker) regions representing states with higher values.

domain serves to reemphasize that basis construction is dependent on a good choice of a local distance metric.

7.2 RPI with Off-Policy Sampling

In the first set of experiments, we used off-policy random walks in Step 1 of the sample collection phase in the RPI algorithm since we wanted to compare the effects of different parameter choices (graph operator, number of nearest neighbors, number of bases) using the *same* set of samples. In Section 7.4 we will see that significantly better results were obtained using a modified form of on-policy sampling. Table 1 summarizes the range of parameters over which the RPI algorithm was tested in these domains. The results for the following experiments were (median) averaged over 30 runs. To avoid clutter, variances are shown only on selected plots.

As Table 1 reveals, the type of off-policy sample collection used in the experiments below varied, from a long series of short random walks (inverted pendulum) to a short series of long random walks (Acrobot). In particular, in the inverted pendulum, samples were collected using a series of short random walks, typically of length < 20 before the episode terminated because the pole was dropped. This simple strategy was sufficient to explore the underlying manifold. By contrast, in the mountain car domain, longer random walks were needed to explore the manifold. 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 mountain car domain are distributed across a wider region of the state space. Finally, in the Acrobot domain, the random walks were very long, terminating when the goal state was reached.

Another difference in sample collection in these domains was in initialization. In the inverted pendulum and Acrobot domains, the initial state was always set the same, with the pole starting from the vertical position at rest, or the arm 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. The experiments reported below scaled the raw state variables to make the dimensions of each variable more commensurate. The scaling used is shown in Table 1.

While performance in all three domains is measured by the number of steps, note that for the Acrobot and mountain car task, lower numbers indicate better performance since we are measuring the steps to reach the goal. In the inverted pendulum, however, since we are measuring the number of steps that the pole remained upright, higher numbers indicate better performance.

Local Distance Metric: In the first experiment, illustrated in Figure 21, the effect of varying the local distance metric used in constructing the graph Laplacian was evaluated, from a low setting of $k = 10$ nearest neighbors to a higher setting of $k = 50$ nearest neighbors. All the plots in the figure show median-averaged plots over 30 learning runs. Variances are not shown to avoid clutter. The effect of varying k was most pronounced in the inverted pendulum domain, with less tangible results in the mountain car and Acrobot domains. Note that in the inverted pendulum domain, the differences between $k = 25$ and $k = 50$ are negligible, and the corresponding runs tightly overlap.

Number of Basis Functions: Figure 22 varied the number of proto-value functions used. Here, there were significant differences, and the results reveal a nonlinear relationship between the number of PVFs used and the best performance. In the Acrobot task, the best results were obtained for 25 and 100 PVFs, and significantly poorer results for 50 PVFs. In the inverted pendulum domain, 10 PVFs was significantly better than using 30 PVFs, but was closely matched by using 60 PVFs. Finally, in the mountain car domain, 30 PVFs produced the best results, followed by 50 PVFs and a setting of 10 PVFs produced the worst results.

Type of Graph Operator: Figure 23 investigates the effect of varying the graph operator in the three domains. The two operators compared were the normalized Laplacian $\mathcal{L} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$ and the combinatorial Laplacian $L = D - W$. In both the Acrobot and mountain car domains, the normalized Laplacian operator produced significantly better results than the combinatorial Laplacian. However, in the inverted pendulum domain, the combinatorial Laplacian was better than the normalized Laplacian operator. These results suggest an interesting dependence between the graph operator and the type of manifold. Note that in both the Acrobot and mountain car domains, the manifold is significantly more spread out spatially than the inverted pendulum task.

7.3 RPI with On-Policy Sampling

As noted earlier, the performance of PVFs can be improved using a modified form of on-policy sampling in Step 1 of the sample collection phase in the RPI algorithm. Specifically,

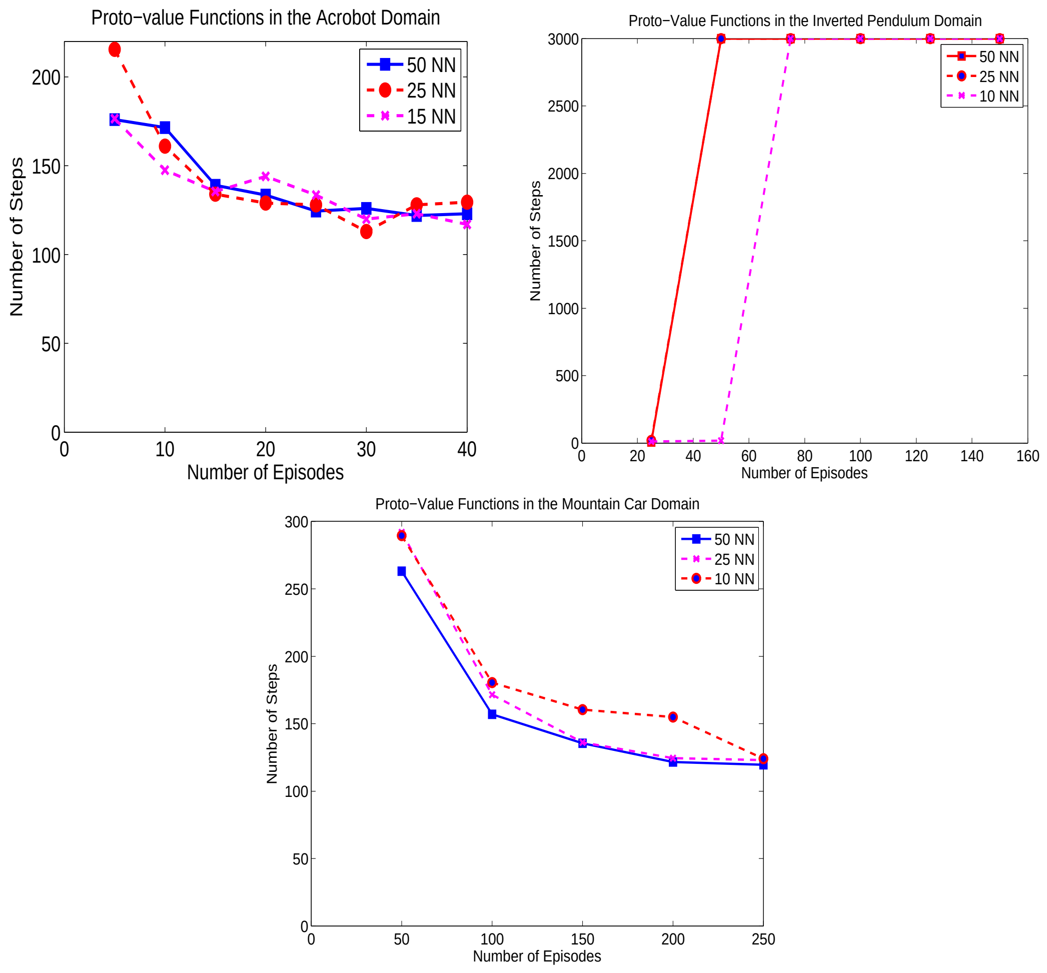


Figure 21: Performance of PVFs on the Acrobot, inverted pendulum, and mountain car domains as a function of the number of nearest neighbors used to compute the graph Laplacian. Results are median averages over 30 learning runs. In all three domains, the graph operator used was the normalized Laplacian. For the Acrobot domain, the number of PVFs was set at 100, whereas in the mountain car and inverted pendulum tasks, the number of PVFs was set to 30.

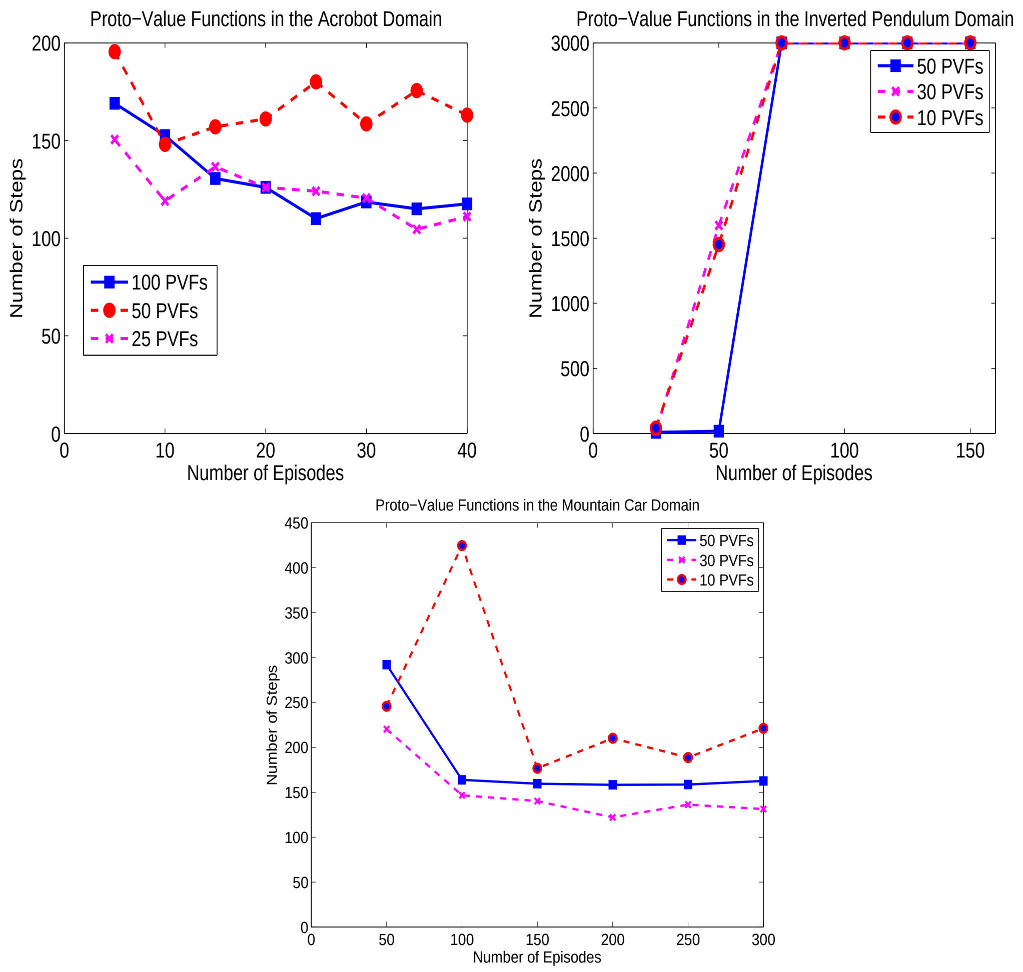


Figure 22: Performance of PVFs on the Acrobot, inverted pendulum, and mountain car domains as a function of the number of basis functions. Results shown are median averages over 30 learning runs. In all three domains, the normalized Laplacian was used as the graph operator. The number of nearest neighbors $k = 25$ in the Acrobot and inverted pendulum domains, and $k = 30$ in the mountain car domain.

we kept track of the best-performing policy (in terms of the overall performance measure of the number of steps). If the policy learned in the current round of RPI improved on the best-performing policy thus far, samples were collected in the next iteration of RPI using the newly learned policy (which was then viewed as the best performing policy in subsequent runs). Otherwise, samples were collected using an off-policy random walk. We also found that using shorter episodes of sample collection in between rounds of representation construction and policy estimation also produced better results. Figure 24 shows the results of these two modifications in the Acrobot domain, whereas Figure 25 and Figure 27 show the corresponding results from the inverted pendulum and mountain car domains.

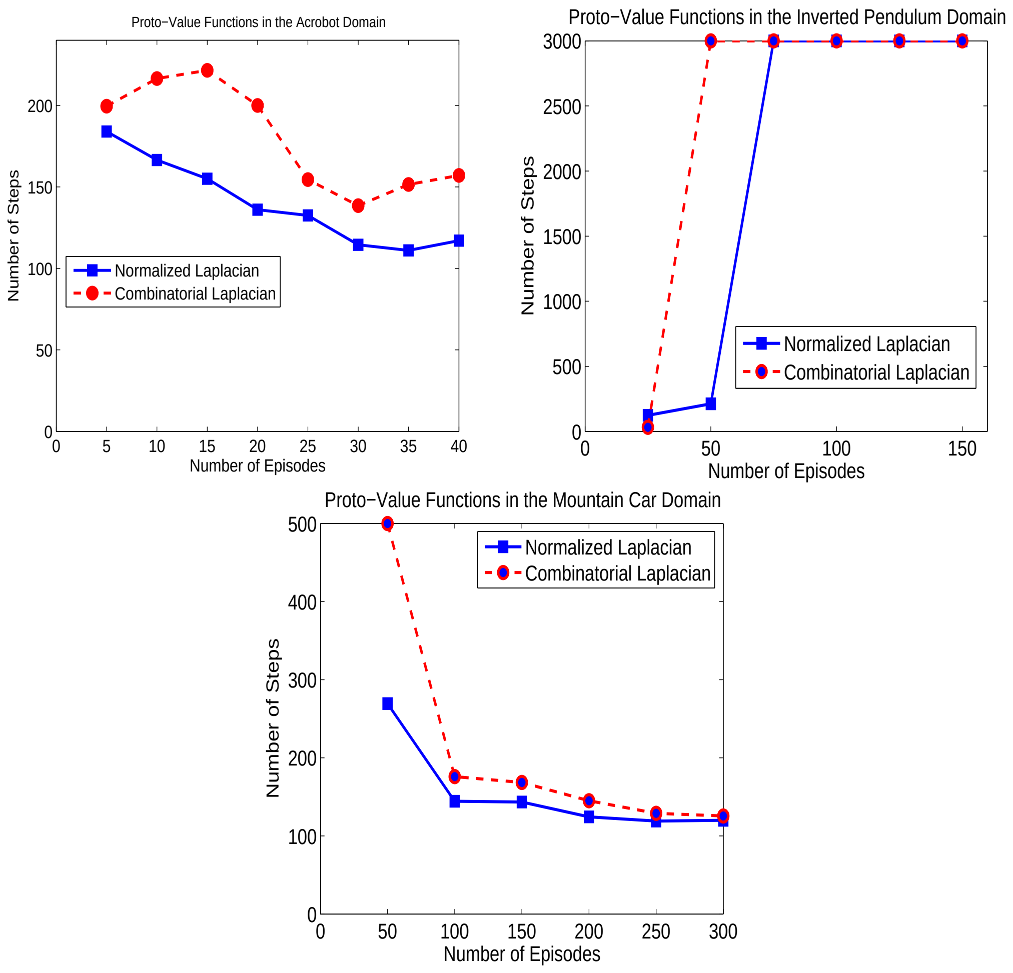


Figure 23: Performance of PVFs in the Acrobot, inverted pendulum, and mountain car domains as a function of the graph operator. Results shown are median averages over 30 learning runs. In the Acrobot task, 100 PVFs were used, whereas 30 basis functions were used in the mountain car task, and 10 basis functions were used in the inverted pendulum task.

Comparing these results with the corresponding off-policy results in Figure 21, Figure 22, and Figure 23 shows significantly faster convergence of PVFs in all three domains.

7.4 Comparing PVFs with RBFs on Continuous MDPs

In this section, we compare the performance of PVFs with radial basis functions (RBFs), which are a popular choice of basis functions for both discrete and continuous MDPS. We restrict our comparison of PVFs and RBFs in this section to the inverted pendulum and mountain car domains. To choose a suitable set of parameters for RBFs, we initially relied on the values chosen in the published study of LSPI for the inverted pendulum domain

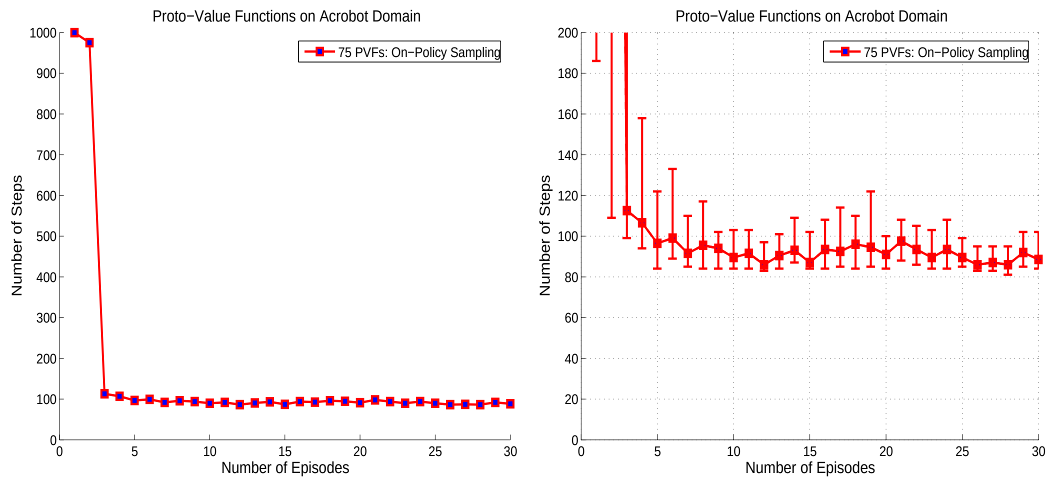


Figure 24: Performance of PVFs with on-policy sampling in the Acrobot task. The plot on the left shows the median average number of steps to goal averaged over 30 runs. The plot on the right shows the variance, after scaling the y axis to magnify the plot.

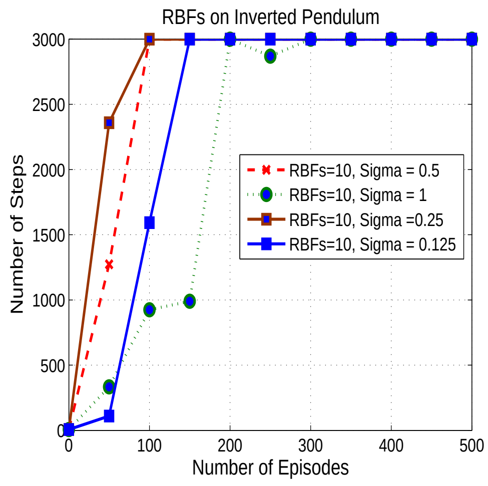
(Lagoudakis and Parr, 2003). However, we found that by tuning the kernel widths, we were able to significantly improve the performance of RBFs over that previously reported in their experiments. Table 2 shows the parameters of the RBF used in the comparisons below. Generally speaking, the results demonstrate that PVFs are significantly quicker to converge, by almost a factor of two in both the inverted pendulum and mountain car domains. Asymptotically, both approaches to converge to the same result. We emphasize that these comparisons are meant to be *suggestive*, and not definitive. For example, we did not fine tune the centers of the RBF bases, or incorporate the scaling factors used in the experiments with PVFs. Our goal here is to provide a reasonable set of benchmarks to compare PVFs against, commensurate with that shown in earlier studies using such parametric approximators.

Inverted Pendulum: We begin by comparing the performance of PVFs with a linear RBF approximation architecture for the inverted pendulum domain. Figure 25 plots the effect of varying the kernel width for RBFs in the inverted pendulum domain (left plot). It is seen that the best results are obtained for a kernel width $\sigma = 0.25$. We compare a varying number of RBF architectures with using 15 PVFs in Figure 25 (right plot). PVFs converge significantly faster to the final goal of balancing the pendulum for 3000 steps: PVFs take 20 trials to converge, but RBFs take roughly twice as long. Figure 26 plots the variance across 100 learning runs for both PVFs and RBFs, showing that PVFs not only converge faster, but also have significantly less variance.

Mountain Car: As with the inverted pendulum, we were able to improve the performance of RBFs by fine-tuning the kernel width, although the differences are less significant than in the inverted pendulum domain. Figure 27 plots the effect of varying the kernel width

Parameter	Inverted Pendulum	Mountain Car	Acrobot
Episodes T	(20 to 160)	(50 to 300)	(5 to 40)
Episode Length N	≤ 20	≤ 70	≤ 800
Nearest neighbors ω	$\{10, 25, 50\}$	$\{10, 25, 50\}$	$\{25, 50, 100\}$
Number of PVFs k	$\{10, 30, 60\}$	$\{10, 30, 50\}$	$\{25, 50, 100\}$
Graph Operator $\mathcal{O}$	(Norm., Comb.)	(Norm., Comb.)	(Norm., Comb.)
Scaling	$(3\theta, \dot{\theta})$	$(x, 3\dot{x})$	$(\theta_1, \theta_2, 0.5\dot{\theta_1}, 0.3\dot{\theta_2})$

Table 1: Parameter values (as defined in Figure 18) for Acrobot, inverted pendulum and mountain car domains. Comb. and Norm. refer to the combinatorial and normalized Laplacian operators.



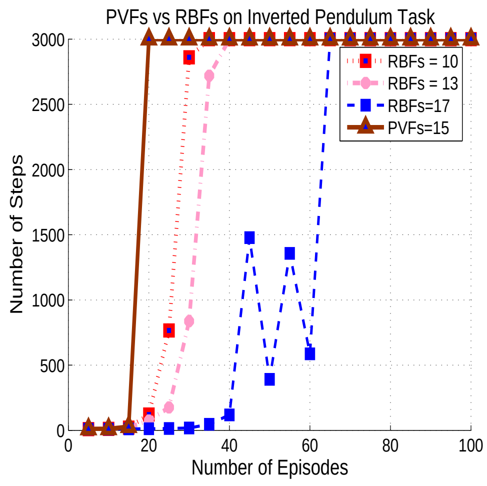


Figure 25: Left: The performance of a linear parametric RBF architecture is analyzed for varying kernel widths in the inverted pendulum domain. Right: A comparison of 15 PVFs with several choices of RBFs on the inverted pendulum task, focusing on the initial 100 episodes averaged over 100 runs.

Number of RBFs	Inverted Pendulum RBF Parameters
10	3 x-axis, 3 y-axis, $\sigma = 1, 0.5, 0.25, 0.125$
13	4 x-axis, 3 y-axis, $\sigma = 0.25$
17	4 x-axis, 4 y-axis, $\sigma = 0.25$
Number of RBFs	Mountain Car RBF Parameters
13	4 x-axis, 3 y-axis, $\sigma = 0.5, 0.1, 0.05$

Table 2: RBF parameter settings for inverted pendulum and mountain car experiments.

for RBFs using 13 basis functions in the mountain car domain (left plot). We also found increasing the number of RBF basis functions above 13 worsened their performance. The figure also plots the best performing RBF architecture (13 basis functions) compared with the PVF approach (25 basis functions). Given sufficient training experience, both converge to approximately the same result, although PVFs seem to converge to a slightly better result. However, as with the inverted pendulum results, PVFs converge significantly quicker, and clearly outperform RBFs for smaller numbers of samples.

Figure 28 shows the variances over 30 runs for both PVFs and RBFs in the mountain car domain. As in the inverted pendulum, we note that PVFs clearly converge more quickly to a more stable performance than RBFs, although the differences are not as dramatic as in the inverted pendulum domain.

8. Related Work

In this section, we briefly review related work, beginning with methods for approximating value functions, followed by a description of past research on representation learning, concluding with a short summary of recent work on manifold and spectral learning.

8.1 Value Function Approximation

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, 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), least squares projection (Bradtke and Barto, 1996; Boyan, 1999; Nedic and Bertsekas, 2003; Lagoudakis and Parr, 2003), and linear programming (de Farias, 2003; Guestrin et al., 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 approximation, including

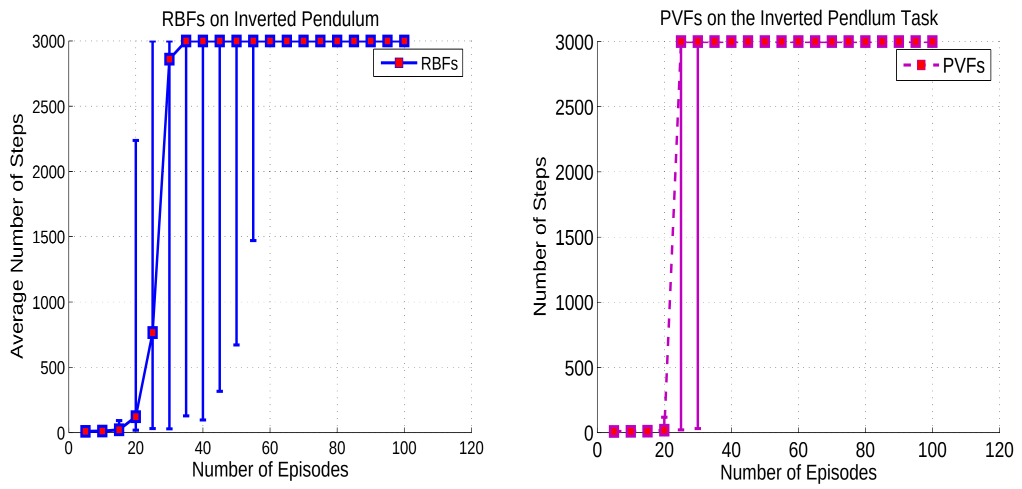


Figure 26: This plot shows that PVFs (right) have significantly less variance compared to RBFs (left) in the inverted pendulum task. Both plots show median-averaged number of steps the pole was balanced over 100 learning runs.

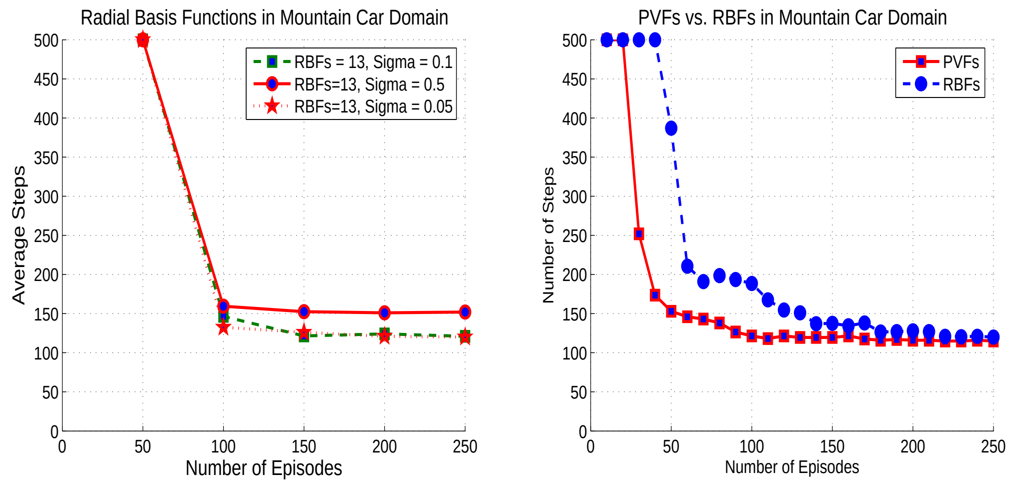


Figure 27: Left: The performance of a linear parametric RBF architecture is analyzed for varying kernel widths in the mountain car domain. Right: A comparison of 25 PVFs and 13 RBFs on the mountain car task. Higher number of RBFs produced worse results.

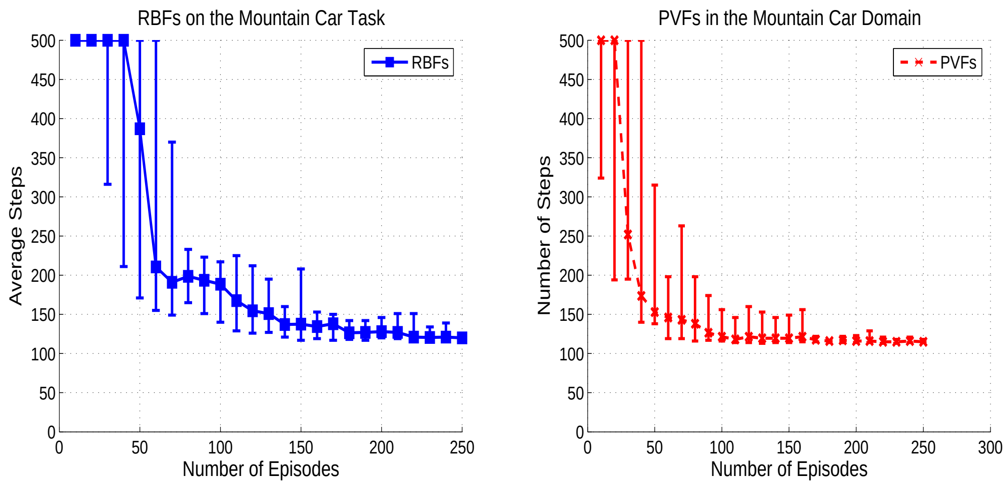


Figure 28: Left: The variance in performance of a linear parametric RBF architecture is analyzed over 30 learning runs in the mountain car domain. Right: Variance across 30 runs for PVFs in the mountain car task.

support vector machines (Dietterich and Wang, 2002) and Gaussian processes (Engel et al., 2003; 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). There are interesting connections between the graph Laplacian matrix and covariance matrices (Ben-Chen and Gotsman, 2005).

8.2 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 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.

8.3 Manifold and Spectral Learning

This research also builds on recent work on manifold and spectral learning, including diffusion maps (Coifman et al., 2005a,b,c), ISOMAP (Tenenbaum et al., 2000), LLE (Roweis and Saul, 2000), and Laplacian eigenmaps (Belkin and Niyogi, 2004; Jones et al., 2007). One major difference is that these methods have largely (but not exclusively) been applied to nonlinear dimensionality reduction and semi-supervised learning on graphs, 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.

9. Discussion and Future Research

The fundamental contribution of this paper is an algorithmic framework called RPI that combines the learning of representations (basis functions) and policies. RPI is based on some specific design choices, and we have naturally restricted our description of the framework to the simplest settings. The scope of RPI can easily be extended to more general situations. Many extensions of the framework are being actively explored, and we briefly summarize these ongoing investigations.

9.1 Analysis of RPI and Variants

RPI is based on a two-phased procedure, where basis functions are learned from spectral analysis of trajectories generated by simulating policies, and improved policies are found by a control learning algorithm using the newly generated basis functions. Section 7 evaluated both the *off-policy* setting, where basis functions were learned purely from random walks, as well as the *on-policy* setting, where additional samples were generated from newly learned improved policies and combined with the random-walk samples. In both approaches, a smaller subset of samples were extracted using a subsampling method described in Section 6.3. Many questions remain to be addressed about the specific properties of architectures like RPI as well as other related architectures that combine the learning of representation and behavior. We summarize some key issues that need to be addressed in future research:

- How can we modify the design of RPI, so that basis functions are learned *simultaneously* with the learning of policies? Recent work on Bellman-error basis functions (Keller et al., 2006; Petrik, 2007; Parr et al., 2007) suggests an alternative approach where basis functions are learned *in-situ* during the policy evaluation phase itself, by explicitly modeling the error in approximating the value function using the Bellman residual. In such approaches, the basis functions generated are very sensitive to a specific reward function, whose shapes reflect the error in approximating a given value function. Can such in-situ basis-function learners be combined with offline approaches such as RPI, where basis functions are generated using a more global analysis of the state space as a whole, to yield more robust provably optimal control learners? For example, Petrik (2007) proposes combining reward-specific Krylov bases with Laplacian bases as a way of integrating localized high-frequency reward-specific bases with more global long-term eigenvector bases such as PVFs. We discuss below other approaches for integrating local vs. global basis functions, such as diffusion wavelets.

- Is it possible to specify optimality metrics for basis function generation, similar to metrics used in control learning such as maximizing the cumulative long-term discounted sum of rewards (or average reward)? How can the cost of learning basis functions be amortized over multiple problems? Does this tradeoff suggest a way to balance the learning of reward-based and reward-independent basis functions?

- What are the pros and cons of off-policy sampling vs. on-policy sampling in designing the outer loop of RPI? For example, is it possible to construct problems where on-policy sampling results in oscillation, as samples are increasingly generated from policies that visit increasingly restricted portions of the state space? In the experiments in Section 7, newly generated samples are combined with previously generated samples to avoid overfitting basis functions to narrow regions of the state space, but this strategy may be computationally expensive in large MDPs.

- Under what assumptions can RPI be shown to converge? It is clear from the experiments presented in Section 7 that RPI converges extremely quickly in problems like the inverted pendulum, whereas in other problems such as the mountain car or Acrobot, convergence takes significantly longer. Can we characterize more formally conditions on the underlying state (action) manifold under which RPI can be shown to reliably converge?

9.2 Combining Nonparametric Graph-based and Parametric Basis Functions

Proto-value functions are given information about the underlying state space manifold in terms of the underlying graph that captures non-local smoothness, whereas parametric bases generally make fairly broad uniformity assumptions about the underlying state space topology. It is reasonable to try to combine the graph-based approach with parametric methods, such as RBFs, to combine the advantages of the two approaches. For example, geodesic Gaussian kernels (Sugiyama et al., 2007) are based on learning a graph of the underlying MDP from random walks, and using the shortest path between any two states as the distance metric for a set of RBFs defined on the graph. The Gaussian exponential term in the RBF approximator can be shown to be the solution of a *diffusion kernel* (Kondor and Lafferty, 2002) or *heat kernel* (Chung, 1997) defined by a differential equation, whose solution can be expressed as a matrix exponential function of the graph Laplacian. Interestingly, matrix exponentials can serve as *generators* of manifold structures called *Lie groups* (Baker, 2001), of which some interesting varieties are rotation and motion groups discussed in more detail in Section 9.8. The Laplacian can also be viewed as an inverse covariance matrix (Ben-Chen and Gotsman, 2005), defining a smoothing prior on the space of functions, which can be contrasted with other priors such as Gaussian processes (Rasmussen and Kuss, 2004; Rasmussen and Williams, 2006). It is possible to combine the graph Laplacian smoothness functional with other parametric smoothing kernels using manifold regularization methods (Belkin et al., 2006).

9.3 Proto-Value Functions From Directed Graphs

In this paper, we constructed PVFs by diagonalizing a symmetric diffusion operator on an undirected graph. This approach can be readily generalized to more elaborate diffusion models which capture asymmetry of actions using *directed* graphs. In particular, PVFs can be constructed by diagonalizing the directed graph Laplacian (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 consequently the leading eigenvector associated with the largest (real) eigenvalue must have all positive components $\phi(v) > 0$. In an initial study (Johns and Mahadevan, 2007), we have found that the directed graph Laplacian can result in a significant improvement over the undirected Laplacian in some discrete and continuous MDPs. For example, in a modified two-room task where there are two "one-way" doors leading from one room to the other, PVFs constructed from the directed Laplacian significantly outperformed the non-directional PVFs constructed from undirected graphs for certain locations of the goal state (e.g., near one of the one-way doors). Directed PVFs also appeared to yield improvements in some continuous control tasks, such as the inverted pendulum.

9.4 Scaling PVFs by Kronecker Product Factorization

Proto-value functions can be made more compact using a variety of sparsification methods, some of which have been explored in the literature on kernel methods. These include matrix sparsification techniques (Achlioptas et al., 2002), low-rank approximation techniques

(Frieze et al., 1998), graph partitioning (Karypis and Kumar, 1999), and Kronecker product approximation (Van Loan and Pitsianis, 1993). We discuss one specific approach that we have implemented for continuous MDPs, and that has given us promising results (Johns et al., 2007). A random walk weight matrix $P_r = D^{-1}W$ constructed through the methods specified above in Section 6 can be approximated by a Kronecker product of two smaller stochastic matrices P_a and P_b, which minimizes the Frobenius norm of the error:

$$f(P_a, P_b) = \min\left(\|P_r - P_a \otimes P_b\|_F\right).$$

We have implemented the approach specified in Van Loan and Pitsianis (1993) to construct two smaller stochastic matrices whose Kronecker product approximates the original random walk matrix P_r. [20] To ensure that the decomposed matrices are not only stochastic, but also diagonalizable, which the Kronecker factorization procedure does not guarantee, we incorporate an additional step using the Metropolis Hastings algorithm (Billera and Diaconis, 2001) to make the smaller matrices P_a and P_b *reversible*. Then, the PVFs for the original random walk matrix P_r can be approximated as the Kronecker product of the PVFs of the factorized smaller reversible matrices P_a^r and P_b^r (since the smaller matrices are reversible, they can also be symmetrized using the normalized Laplacian, which makes the numerical task of computing their eigenvectors much simpler). In an initial study (Johns et al., 2007), we have been able to significantly reduce the size of the random walk weight matrices for the inverted pendulum, mountain car, and the Acrobot tasks with modest loss in performance compared to the full matrix. For example, in the Acrobot task, the original basis matrix is compressed by a factor of 36 : 1, which resulted in a policy slightly worse than the original larger basis matrix. One important point to emphasize is that the full basis matrix never needs to be stored or computed in constructing the state embeddings from the smaller matrices. The factorization can be carried out recursively as well, leading to a further reduction in the size of the basis matrices.

9.5 Multiscale Diffusion Wavelet Bases

In this paper, proto-value functions were constructed by diagonalization, that is by finding eigenvectors, of a symmetrized diffusion operator such as the Laplacian on an undirected graph. Formally, such eigenvectors are essentially global *Fourier* bases and their properties have been extensively studied in Euclidean spaces (Mallat, 1989). One well-known limitation of global Laplacian bases is that they are poor at representing piecewise linear (value) functions. We have extended the approach presented in this paper to construct multiscale diffusion bases, using the recently proposed *diffusion wavelet* framework (Coifman and Maggioni, 2006; Bremer et al., 2006). Diffusion wavelets provide an interesting alternative to global Fourier eigenfunctions for value function approximation, since they encapsulate all the traditional advantages of wavelets (Mallat, 1989): 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 Mahadevan and Maggioni

20. It is important to distinguish this approach from the Kronecker decomposition approach described in Section 5, where the factorization was not an approximation, but an exact decomposition assuming the overall state space was a product space. Here, the Kronecker factorization can be applied to arbitrary weight matrices, but the decomposition is an approximation.

(2006) we compare the performance of diffusion wavelet bases and Laplacian bases on a variety of simple MDPs. In Maggioni and Mahadevan (2006), we present an efficient direct method for policy evaluation by using the multiscale diffusion bases to invert the Bellman matrix $I - \gamma P^\pi$. We are currently exploring faster methods of constructing multiscale diffusion wavelet bases.

9.6 Policy and Reward-Sensitive PVFs

In the PVF framework presented above, basis functions are constructed without taking rewards into account. This restriction is not intrinsic to the approach, and reward or policy information when available can easily be incorporated into the construction of PVFs. One recent approach studied in Petrik (2007) assumes that the reward function R^π and policy transition matrix P^π are known, and combines Laplacian PVF bases with *Krlyov bases*. This approach is restricted to *policy evaluation*, which consists of solving the system of linear equations

$$(I - \gamma P^\pi)V^\pi = R^\pi.$$

This equation is of the well-studied form $Ax = b$, and Krylov bases are used extensively in the solution of such linear systems of equations. The Krylov space is defined as the space spanned by the vectors

$$\left(b\ Ab\ A^2b\ \ldots A^{m-1}b\right).$$

The use of Krylov bases to compress the *belief* space of a partially-observable Markov decision process (POMDP) is investigated in Poupart and Boutilier (2003), which explores how to exploit the factored representation of the transition dynamics specified by a dynamic Bayes net. As discussed earlier, Keller et al. (2006) and Parr et al. (2007) both investigate constructing reward-sensitive basis functions by explicitly estimating the error in approximating the value function using the *Bellman residual*. These approaches can also be combined with Laplacian PVFs in several ways, for example by combining low-frequency Laplacian bases with the more high-frequency reward-specific Krylov bases, or by using the estimated Bellman residuals to set the weights of the graph.

A more direct way to incorporate reward-sensitive information into PVFs is to modify the weight matrix W to take into account the *gradient* of the value function to be approximated. Formally, this approach is similar to estimating a function by knowing not only its values at sample points, but also its gradient. Of course, any errors in the estimation of such gradients will then be reflected in the weight matrix, and such an approach is not also without some drawbacks. While making bases sensitive to rewards can lead to superior results, if the reward function or policy is modified, reward-sensitive basis functions would need to be re-learned. In comparison, reward-independent bases may be more generally applicable across different tasks.

9.7 Learning State Action PVFs

In our paper, the basis functions $\phi(s)$ are originally defined over states, and then extended to state action pairs $\phi(s, a)$ by duplicating the state embedding $|A|$ times and "zeroing" out elements of the state-action embedding corresponding to actions not taken. That is, $\phi(s,a) = \phi(s) \otimes I_a$ where I_a is a vector indicator function for action a (all elements of I_a are 0 except for the chosen action). This construction is somewhat wasteful, especially in domains where the number of actions can vary significantly from one state to another. We have recently implemented PVFs on *state action* graphs, where vertices represent state action pairs. Thus, the pair (s,a) is connected by an edge to the pair (s',a') if action a in state s resulted in state s' from which action a' was next attempted. State action graphs are naturally highly directional, and we used the directed Laplacian to compute basis functions over state action graphs. Our initial results (Osentoski and Mahadevan, 2007) show that state action bases can significantly improve the performance of PVFs in discrete MDPs.

9.8 Group-Theoretic Methods for Constructing Proto-Value Functions

As we discussed earlier in Section 3.6, there is a long tradition in mathematics of constructing representations that are invariant under a group operator, including Fourier and wavelet transforms (Mallat, 1989). One interesting extension is to exploit the properties of linear (matrix) representations of groups to construct compact PVFs. In particular, many of the continuous MDPs we studied, including the inverted pendulum and the Acrobot, define continuous manifolds that have been extensively studied in mathematics (Baker, 2001) and robotics (Lavalle, 2006). In addition, the product spaces described in Section 5 generate graphs with large automorphism groups, which can be exploited in reducing the size of their associated Laplacian eigenspaces.

To make this more concrete, consider the set of points generated by a rotation of a rigid object in $\mathbb{R}^2$. This manifold can be modeled as a *Lie* (matrix) group called $SO(2)$, which stands for special orthogonal group of order 2. This rotation group is defined by all orthogonal matrices whose determinant is 1. Rotations and translations in $\mathbb{R}^2$ can be represented by another Lie group called $SE(2)$ (special Euclidean group). Finally, problems like the Acrobot task are instances of *kinematic chains*, which can be modeled by products of $SE(2)$ matrices. These groups generalize correspondingly to higher dimensions. Note that $SE(n)$ groups are non-Abelian because rotations do not commute with translations—the order matters! A detailed overview of Fourier analysis on non-Abelian groups is given in Chirikjian and Kyatkin (2001), with an emphasis on rotation and motion groups useful in robotics. An interesting direction for future work is to exploit such group representations to construct compact PVFs.

9.9 Proto-Value Functions for Semi-Markov Decision Processes

Proto-value functions provide a way of constructing function approximators for hierarchical reinforcement learning (Barto and Mahadevan, 2003), as well as form a theoretical foundation for some 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; Şimşek et al., 2005). In particular, Şimşek et al. (2005) use the second eigenvector of the discrete graph Laplacian operator $I - D^{-1}W$ to find bottlenecks in (undirected) state space graphs. Ravindran and Barto (2003) explore the use of group homomorphisms on state action spaces to abstract semi-MDPs, which can be combined with PVFs as a way of solving large SMDPs.

Another direction that we have begun exploring is to construct PVFs for temporally extended actions, such as “exiting a room”. These temporally extended actions result in longer “distal” edges connecting non-adjacent vertices (such as the vertices corresponding to interior states in a room with those representing the “door” state). Our initial results reported in Osentoski and Mahadevan (2007) suggest that constructing PVFs over state-action graphs using these distal edges can significantly improve the performance over PVFs constructed over state graphs with only primitive actions.

9.10 Theoretical Analysis

Theoretical guarantees on the efficiency of proto-value functions in approximating value functions are being investigated. 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 on the subspace spanned by the smallest k proto-value functions produces *globally* the smoothest approximation taking the graph or manifold into account (Mahadevan and Maggioni, 2006). 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. Belkin and Niyogi (2005) and Hein et al. (2007) study the sampling conditions under which the various graph Laplacians converge to the Laplace-Beltrami operator on the underlying manifold. For example, Hein et al. (2007) show that under non-uniform sampling conditions, the random walk Laplacian converges to a weighted Laplace-Beltrami operator. These results need to be combined with exploration techniques to investigate the conditions under which these sampling conditions can be met in the context of MDPs. 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; Sato, 1995), which quantifies the degree to which small perturbations of (positive definite) matrices lead to bounded changes in the spectrum and eigenspace as well.

9.11 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 suggest a solution to the well-known problem of transfer in reinforcement learning (Mahadevan, 1992; Sherstov and Stone, 2005). One key advantage of proto-value functions is that they provide a theoretically principled approach to transfer, which respects the underlying state (action) space manifold. We have recently begun to investigate a framework called *proto-transfer* learning to explore the transfer of learned representations from one task to another (in contrast to transferring learned policies) (Ferguson and Mahadevan, 2006).

10. Summary

This paper describes a novel spectral framework for learning both representation and control in Markov decision processes, where basis functions called proto-value functions are constructed by diagonalization of a symmetric diffusion operator learned from samples collected during a random walk of the underlying state space. Proto-value functions can be defined in several ways: this paper focused principally on using the graph Laplacian on undirected graphs. Eigenfunctions of the graph Laplacian provide geometrically customized basis functions that capture large-scale properties such as bottlenecks and symmetries. Projections of a value function onto the eigenfunctions of the graph Laplacian provide the globally smoothest approximation that respects the underlying graph or manifold. A general algorithmic framework called representation policy iteration (RPI) was presented consisting of three components: sample collection, basis function construction, and control learning. A specific instance of RPI was described that uses the least-squares policy iteration (LSPI) method as the underlying control learner. Several directions for scaling the approach were described, including Kronecker sum matrix factorization for large factored MDPs, and sparse sampling combined with the Nystroṁ interpolation method for continuous MDPs. Detailed experimental results were provided using benchmark discrete and continuous MDPs, which evaluated the effectiveness of the proto-value function approach, and compared their performance to handcoded parametric function approximators, such as polynomials and radial basis functions. Many extensions of the proposed framework are possible, and a few promising directions were elaborated.

Acknowledgments

We are indebted to the anonymous reviewers and the action editor for their detailed comments on an earlier draft of this paper. The first author would like to thank members of the PVF group—Kimberly Ferguson, Jeff Johns, Vimal Mathew, Sarah Osentoski, Marek Petrik, Ilya Scheidwasser, Andrew Stout, and Chang Wang—for their valuable input. We are also grateful to Kimberly Ferguson, Jeff Johns, and Sarah Osentoski for carrying out some of the experiments. We would like to thank Andrew Barto and other members of the Autonomous Learning Laboratory for their feedback. Support for this research was provided in part by the National Science Foundation under grants IIS-0534999 and DMS-0650413.

References

D. Achlioptas, F. McSherry, and B. Scholkopff. Sampling techniques for kernel methods. In *Proceedings of the* 14*th* *International Conference on Neural Information Processing Systems (NIPS)*, pages 335–342. MIT Press, 2002.

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.

J. Bagnell and J. Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1019–1024, 2003.

A. Baker. *Matrix Groups: An Introduction to Lie Group Theory.* Springer, 2001.

C. 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 (COLT)*, pages 486–500, 2005.

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

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

S. Belongie, C. Fowlkes, F. Chung, and J. Malik. Spectral partitioning with indefinite kernels using the Nyström extension. In *Proceedings of the 7^{th} European Conference on Computer vision*, pages 531–542, 2002.

M. Ben-Chen and C. Gotsman. On the optimality of spectral compression of mesh data. *ACM Transactions on Graphics*, 24(1), 2005.

A. Bernasconi. *Mathematical Techniques for Analysis of Boolean Functions.* PhD thesis, University of Pisa, 1998.

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

L. Billera and P. Diaconis. A geometric interpretation of the Metropolis-Hasting algorithm. *Statistical Science*, 16:335–339, 2001.

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

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.

J. Bremer, R. Coifman, M.Maggioni, and A. Szlam. Diffusion wavelet packets. *Applied and Computational Harmonic Analysis*, 21(1):95–112, July 2006.

G. Chirikjian and A. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis.* CRC Press, 2001.

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. *Spectral Graph Theory.* Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

F Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, April 2005.

F. Chung and S. Sternberg. Laplacian and vibrational spectra for homogeneous graphs. *Journal of Graph Theory*, 16(6):605–627, 1992.

R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, July 2006.

R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data. part i: Diffusion maps. *Proceedings of National Academy of Science.*, 102(21):7426–7431, May 2005a.

R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, Frederick Warner, and Steven Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data. part ii: Multiscale methods. *Proceedings of the National Academy of Science*, 102 (21):7432–7437, May 2005b.

R. Coifman, M. Maggioni, S. Zucker, and I. 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.

D. Cvetkovic, P. Rowlinson, and S. Simic. *Eigenspaces of Graphs.* Cambridge University Press, 1997.

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

D. de 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.

F. 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.

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

Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 154–161. AAAI Press, 2003.

K. Ferguson and S. Mahadevan. Proto-transfer learning in Markov decision processes using spectral methods. In *International Conference on Machine Learning (ICML) Workshop on Transfer Learning*, 2006.

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.

A. Graham. *Kronecker Products and Matrix Calculations: With Applications.* Ellis Horwood, 1981.

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.

D. Gurarie. *Symmetries and Laplacians: Introduction to Harmonic Analysis, Group Representations and Laplacians.* North-Holland, 1992.

M. Hein, J. Audibert, and U. von Luxburg. Graph Laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

J. Jackson. *The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice.* PhD thesis, Carnegie-Mellon University, 1995.

J. Johns and S. Mahadevan. Constructing basis functions from directed graphs for value function approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 385–392. ACM Press, 2007.

J. Johns, S. Mahadevan, and C. Wang. Compact spectral bases for value function approximation using Kronecker factorization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2007.

P. Jones, M. Maggioni, and R. Schul. Universal parametrizations via eigenfunctions of the Laplacian and heat kernels. Submitted, 2007.

S. Kakade. A Natural Policy Gradient. In *Proceedings of Neural Information Processing Systems.* MIT Press, 2002.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1999.

P. Keller, S. Mannor, and D Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 22^{nd} International Conference on Machine Learning (ICML)*, pages 449–456. MIT Press, 2006.

D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proceedings of the 16th Conference on Uncertainty in AI*, pages 326–334, 2000.

R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the 19th International Conference on Machine Learning*, pages 315–322, 2002.

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

R. Kretchmar and C. Anderson. Using temporal neighborhoods to adapt function approximators in reinforcement learning. In *International Work Conference on Artificial and Natural Neural Networks*, pages 488–496, 1999.

S. Kveton and M. Hauskrecht. Learning basis functions in hybrid domains. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2006.

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

S. 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*. Cambridge University Press, 2006.

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

M. Maggioni and S. Mahadevan. Fast direct policy evaluation using multiscale analysis of Markov Diffusion Processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 601–608, New York, NY, USA, 2006. ACM Press.

S. Mahadevan. Proto-Value Functions: Developmental Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, pages 553–560, 2005a.

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 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 372–37. AUAI Press, 2005b.

S. Mahadevan and M. Maggioni. Value function approximation with Diffusion Wavelets and Laplacian Eigenfunctions. In *Proceedings of the Neural Information Processing Systems (NIPS)*. MIT Press, 2006.

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

S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):674–693, 1989. ISSN 0162-8828.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *International Conference on Machine Learning*, 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 *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1006–1011, 2005.

R. Munos. Error bounds for approximate policy iteration. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 560–567, 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. In *NIPS*, 2002.

P. Niyogi, I. Matveeva, and M. 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.

S. Osentoski and S. Mahadevan. Learning State Action Basis Functions for Hierarchical Markov Decison Processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 705–712, 2007.

R. Parr, C. Painter-Wakefiled, L. Li, and M. Littman. Analyzing feature generation for value function approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 737–744, 2007.

R. Patrascu, P. Poupart, D. Schuurmans, C. Boutilier, and C. Guestrin. Greedy Linear Value Function Approximation for Factored Markov Decision Processes. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 285–291, 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.

M. Petrik. An analysis of Laplacian methods for value function approximation in MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2574–2579, 2007.

P. Poupart and C. Boutilier. Value directed compression of POMDPs. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2003.

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*, pages 751–759. MIT Press, 2004.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

B. Ravindran and A. Barto. SMDP homomorphisms: An algebraic approach to abstraction in Semi-Markov Decision Processes. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 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.

T. Sato. *Perturbation Theory for Linear Operators.* Springer, 1995.

B. Scholkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

A. Sherstov and P. Stone. Improving action selection in Markov Decision Processes 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.

O. Şimşek, A. Wolfe, and A. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 816–823, 2005.

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

D. Subramanian. *A Theory of Justified Reformulations.* Ph.D. Thesis, Stanford University, 1989.

M. Sugiyama, H. Hachiya, C. Towell, and S. Vijaykumar. Value function approximation on non-linear manifolds for robot motor control. In *Proceedings of the IEEE Conference on Robots and Automation (ICRA)*, 2007.

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

A. Szlam, M. Maggioni, and R. Coifman. A general framework for adaptive regularization based on diffusion processes on graphs. Technical Report YALE/DCS/TR1365, Yale Univ, July 2006.

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.

M. Thornton, R. Drechsler, and D. Miller. *Spectral Methods for VLSI Design.* Kluwer Academic, 2001.

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. Van Loan and N. Pitsianis. Approximation with Kronecker products. In *Linear Algebra for Large Scale and Real Time Applications*, pages 293–314. Kluwer Publications, 1993.

B. Van Roy. *Learning and Value Function Approximation in Complex Decision Processes.* PhD thesis, MIT, 1998.

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

C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 682–688, 2000.