Abstract

In reinforcement learning (RL), an important sub-problem is learning the value function, which is chiefly influenced by the architecture used to represent value functions. Often, the value function is expressed as a linear combination of a pre-selected set of basis functions. These basis functions are either selected in an ad-hoc manner or are tailored to the RL task using the domain knowledge. Selecting basis functions in an ad-hoc manner does not give a good approximation of value function while choosing functions using domain knowledge introduces dependency on the task. Thus, a desirable scenario is to have a method to choose basis functions that are task independent, but which also provide a good approximation for the value function. In this paper, we propose a novel task-independent method to construct reward-based Proto Value Functions (RPVFs) using the topology of the state space and the reward structure of the underlying RL task. Our methodology uses the connectivity of the state space and the immediate reward structure to construct the basis functions which are required for linear approximation of the value function. The approach we propose gives enhanced learning performance. In particular, when the state space is symmetrical and the value function asymmetrical, the basis functions so constructed capture the asymmetry in value function better than any of the previous approaches. We demonstrate the effectiveness of RPVFs in approximating the value function via experiments on benchmark RL problems as well as on another non-standard problem.

1 INTRODUCTION

Reinforcement learning (RL) is a sequential decision making paradigm to solve adaptive control problems under model uncertainties. RL problems are cast in the framework of Markov decision processes (MDPs). In the MDP setting, dynamics of the underlying environment evolves within a set of states called the state-space, and the agent performs actions to control the state of the system. Depending on the state and the action the agent performs in that state, the agent receives a reward. The agent aims to maximize the infinite sum of discounted rewards obtained as a result of its actions. Formally, an action selection mechanism is known as a policy and the agent aims to learn an optimal policy.

In order to learn the optimal policy, the agent first needs to evaluate the current policy. The value function $J_u$ corresponding to a given policy $u$, is a map from the state space to real numbers, and captures the total discounted reward that agent collects by following the policy $u$. From the knowledge of the value function, the agent can improve its policy and hence learning the value function in an efficient manner assumes importance.
RL algorithms like Q-learning [1, 31] and its variants [13, 14], SARSA [1] can be used by the agent to learn optimal policies and value function in MDPs with discrete state and action spaces. Q-learning and SARSA store state-action values in a lookup table which are updated using simulation samples of the MDPs. However, when the state and action spaces are large, storing the table consumes a lot of memory and accessing and updating the lookup table is computationally intensive. Thus, these tabular approaches suffer from the lack of a proper representation for the value function. Other learning approaches are policy gradient method [30] and actor-critic [4] methods. In the policy gradient method, the policy is considered as an arbitrary differentiable function of a parameter vector. Given the sample trajectories, an unbiased estimate of the gradient of the value function is constructed and the parameters are updated using this estimate. In the actor-critic method, the critic learns the value function corresponding to a policy determined by the actor and this leads to an improvement in the policy. During the search process, both methods improve policies by evaluating the value function. Thus, the efficiency of policy search methods too depend on the value function representation. Hence, irrespective of the method used, finding a suitable representation for the value function is an important objective.

A desirable property is that the value function representation has to be compact (i.e., it should be easy to compute and store). The linear function approximation is the most widely used, wherein, the value function is represented as a linear combination of basis functions. In general, basis is chosen using task-specific knowledge and when there is no such task-specific information, primitive functions such as tile coded bases [1], Fourier basis [16], polynomial bases [17] and radial basis functions (RBFs) [22] are chosen. However, the important consideration is to formulate a method which selects a compact representation that effectively approximates the value function with little or no information about the task.

Selecting appropriate basis functions (independent of the underlying MDP) and designing a method for optimizing basis functions is still an ongoing work. As previously referred to, hand-coded basis functions do not usually capture the connectivity information which can be exploited to optimally control the underlying MDP. Thus, there is a need for algorithms which build and optimize basis functions by leveraging the geometry of the state space and the reward (or cost) structure. Aiming at this problem, we develop the Shaping-based Representation Policy Iteration (SRPI) algorithm.

In this paper, we present the reward-shaping based graph Laplacian framework. Immediate rewards indicate which transitions are beneficial. Using the knowledge of transitions which yield good and bad rewards, we construct the graph underlying the MDP and the Laplacian [7]. Thus the graph so constructed is sensitive to immediate rewards. Spectral analysis of the Laplacian gives us eigenvectors which are then used as basis functions in state-action value function approximation. Before we proceed, we first review the existing approaches for value function approximation and construction of basis functions.

1.1 Related Work

Recent works have focussed on linear architecture for value function approximation in RL and Approximate Dynamic Programming (ADP). The general trend has been to express the value function as a linear combination of pre-defined basis functions and then to adapt the features. In [22], parameterized RBFs are considered as feature vectors. The parameters of the RBFs are then adjusted using two methods - a gradient based method and Cross Entropy (CE) method. Laurent series expansion of the discounted value function is utilized to construct a Drazin basis representation in [20]. An improved version of the Drazin bases, called as the Bellman average-reward bases was also discussed in [20] and which were shown to converge more quickly when compared to other bases for large values of discount factor.

Moving away from hand-coded basis functions, an automatic basis function construction was proposed in [15]. Here, neighbourhood component analysis is used repeatedly to project the state space onto a lower dimensional space. In the lower dimensional space, aggregation is done to construct basis functions which are then trained using least squares temporal difference (LSTD) learning. In [29] an incremental procedure for expanding the available set of basis functions is proposed. In this work a primary reinforcement learner estimates the value function over a set of basis functions. Further, the temporal difference (TD) error of the underlying MDP is used as reward function to obtain a new value function, which is then used to expand the earlier used set of basis functions. Feature construction in actor-critic methods is considered in [27, 3]. The authors in [27] design state action value function features for both actor and critic using a family of parameterized Gaussian policies. In [3] a different approach of adaptive feature
tuning has been developed to estimate the value function for a discounted cost MDP.

The proto-value functions (PVFs) were introduced in [18]. PVFs are constructed using spectral analysis of the self-adjoint Laplacian operator. The eigenvectors corresponding to the smallest eigenvalues of the Laplacian operator are used as the basis functions. A policy iteration approach (RPI) using the Laplacian was also proposed in [18]. Here, the PVFs were shown to capture the intricate connectivity in the underlying state space which primitive bases such as Fourier or polynomials do not capture. The least squares policy iteration (LSPI) extension to [18] was proposed in [19]. Conditions for the performance of PVFs were analysed in [24], wherein it is also shown that PVFs are related to augmented Krylov methods. Basis function construction using geodesic Gaussian kernels were proposed in [28]. The Gaussian kernels are defined based on the shortest path distance metric and use non-linear manifold structure induced by MDPs. Graph Laplacian in the context of transfer learning was investigated in [10]. The authors propose to approximate value functions in both the source and target domains using PVFs. The weights of the PVFs in the source domain are transferred to the target domain. Two types of domain transfer - scaling and topological were investigated.

Incorporation of rewards in spectral decomposition of the Laplacian was discussed in [8] which uses the concept of bisimulation metrics. These metrics can be iteratively computed and are used to quantify the similarity between states. This similarity is used to form clusters of states. A recent work in Laplacian based framework is [32] which adopts K-means and fuzzy C-means clustering for the subsampling stage. Samples with little information about the underlying MDP graph are filtered out using clustering techniques. The graph Laplacian is then constructed based on the centers of all clusters.

1.2 Our Contribution

In this paper, we present the reward-shaping based graph Laplacian framework. Immediate rewards are used to construct the reward-based diffusion matrix and the Laplacian associated with MDP. The rewards of the underlying MDP are shaped to improve the pace of learning. Reward shaping [5, 6, 23] is the process of providing additional rewards to the learning agent to guide its learning process. We also incorporate risk-sensitivity [9, 12] for rewards in the construction of the diffusion matrix and the Laplacian. Through this construction, we show that the learning agent is prevented from taking actions which lead it to unfavourable states. Using the spectral analysis of the reward-based diffusion matrix, we obtain the reward-based Proto Value Functions (RPVFs). Simulation results and experiments show that the error in value function approximation is minimized when SRPI is used as compared to previous approaches [18, 32]. We illustrate the performance of RPVFs on benchmark RL problems (see Section 4). The characteristic of RPVFs is then explicitly demonstrated in the foraging problem which is an important study in Ecology. Our experiments demonstrate that similarity matrices other than the diffusion matrix can be used to generate features. When the state space is symmetrical and the rewards are asymmetrical, the RPVFs capture the asymmetry better than the PVFs.

1.3 Organisation of the Paper

The paper is organised as follows. In Section 2 we present an overview of the reinforcement learning paradigm, emphasizing the need to learn the value function. This section also introduces the notation which is followed in the later sections of the paper. The reward shaping based graph Laplacian framework is presented in Section 3. In Section 4, we discuss in detail the simulation and experimental results on four problem domains. Section 5 concludes the paper and suggests future enhancements.

2 REINFORCEMENT LEARNING PARADIGM

A Markov decision process (MDP) captures the dynamics of the underlying environment in a RL problem. It is a 4-tuple \(< S, A, P, R >\), where \( S \) is the state space, \( A \) is the action space, \( P \) is the probability transition kernel and \( R \) is the reward function. The probability transition kernel \( P \) specifies the probability \( p_{a}(s, s') \) of transitioning from state \( s \) to state \( s' \) under the action \( a \). The reward function \( R \) is a map \( R: S \times A \rightarrow R \) that specifies the reward obtained for performing action \( a \in A \) in state \( s \in S \) and is denoted by \( r_{a}(s) \).
2.1 Agent Behavior

The behavior of the agent is captured in the actions it chooses in each and every state. In the MDP parlance, this action selection mechanism is called the policy. Formally, by a policy, we mean a sequence \( \mu = \{\mu_0, \ldots, \mu_n, \ldots\} \) of functions \( \mu_i, i \geq 0 \) that describe the manner in which an action is picked in a given state at time \( i \). Two important types of policies that are useful are: (i) Stationary Randomized Policy (SRP), given by \( \mu = \{\mu_0, \ldots, \mu_i, \ldots\} \), where \( \mu_i \equiv \pi, \forall i \geq 0 \) with \( \pi(s, \cdot) \) being a probability distribution over the set of actions for any \( s \in S \). (ii) Stationary Deterministic Policy (SDP), given by \( \mu = \{\mu_0, \ldots, \mu_i, \ldots\} \), where \( \mu_i \equiv u, \forall i \geq 0 \) with \( u : S \to A \) being a map from the state space to the action space. Note that an SDP is trivially an SRP as well. By abuse of notation, we refer to an SRP by \( \pi \) and an SDP by \( u \). Further, under a stationary policy \( u \) (or \( \pi \)), the MDP is a Markov chain and we denote its probability transition kernel by \( P_u = (p_u(i, j), i, j = 1, \ldots, n) \) (or \( P_\pi = (p_\pi(i, j), i, j = 1, \ldots, n) \)), where \( p_\pi(i, j) = \sum_{a \in A} \pi(i, a)p_u(i, j) \) and \( \pi(i) = (\pi(i, a), a \in A) \).

![Figure 1: Different components in the RL paradigm](image)

2.2 Value Function

We define the infinite horizon discounted reward value function under a policy \( \pi \) as \( J^\pi(s) = E \left[ \sum_{t=0}^{\infty} \alpha^t r_t | s_0 = s, \pi \right] \), where \( \alpha \in (0, 1) \) is the discount factor and \( r_t = r_a(s_t) \) with \( a_t \sim \pi(s_t, \cdot) \) for all \( t \geq 1 \). Similarly, we also define the infinite horizon discounted reward state-action value function under a policy \( \pi \) as \( Q^\pi(s, a) = E \left[ \sum_{t=0}^{\infty} \alpha^t r_t | s_0 = s, a_0 = a, \pi \right] \).

The optimal policy\(^1\) and the optimal value function obey the Bellman equation (BE) given below: \( \forall s \in S \),

\[
Q^* (s, a) = (r_a(s) + \alpha \sum_{s'} p_a(s, s') \max_{a' \in A} Q^*(s', a')) ,
\]

\[
u^* (s) = \arg \max_{a \in A} Q^* (s, a).
\]

2.3 Learning Agent

A RL agent has three important building blocks or sub-functions namely sample collection, the representation and the learning algorithm. Learner represents state of the environment \( s_t \) at time \( t \) as a point in the feature space. The learning algorithm makes use of samples \( (s_t, r_t), t \leq n \) obtained from the environment and its own past behavior \( a_t, t \leq n \) to learn.

The behavior of the agent is dictated by policy \( \pi \) it makes use of to choose the actions. From (1) it is clear that in order to compute the optimal behavior \( \nu^* \), the agent needs to learn \( Q^* \). Even in the case when the agent wants to improve a given policy \( \pi \), it has to evaluate \( Q^\pi \) and then substituting \( Q^\pi \) in (1) will lead to an improved policy [2]. Thus, learning the value function is central to learning the correct behavior. Such learning is dependent on the agent’s method of representing the value functions.

\(^{1}\)In the infinite horizon discounted reward setting that we consider, one can find an SDP that is optimal [2, 25]
2.4 Value Function Representation

The most widely used representation is the linear function representation, wherein, the value $Q^\pi(s,a)$ of state-action pair $(s,a)$ is expressed as a weighted combination of the feature corresponding to that state, i.e.,

$$Q^\pi(s,a) = \sum_{i=1}^{k} \phi_i(s,a)^\top w^\pi(i),$$

where $\phi_i(s,a), i = 1, \ldots, k \in \mathbb{R}^k$ is the feature of the state $s$ and $w^\pi \in \mathbb{R}^k$ is a learned weight vector. Any linear representation can be compactly represented by its feature matrix $\Phi = [\phi_1, \ldots, \phi_k]$, where $(\phi_i(s,a), i = 1, \ldots, k) \in \mathbb{R}^k$ is the feature of the state $s$ and $w^\pi \in \mathbb{R}^k$ is a learned weight vector. Classical numerical schemes such as value iteration, policy iteration and linear programming choose a look up table representation. Under the look up table representation the standard basis is chosen, i.e., $\phi_1 = (1, 0, \ldots, 0)^\top$.

Thus there are as many basis functions as the number of state-action pairs and as a result this approach will not be efficient, when the number of state-action pairs is large.

3 REWARD-SHAPING BASED GRAPH LAPLACIAN FRAMEWORK

The building blocks of the graph Laplacian framework we propose are illustrated in Fig. 2. First, we describe how the diffusion matrix is constructed using the trajectory and reward information. Following this, we briefly describe the extension for continuous state spaces. Finally we give the overview of the SRPI algorithm.

![Figure 2: Reward Shaping based graph Laplacian framework](image)

3.1 Constructing the diffusion matrix

Let $G = (E, V)$ denote a graph with edge set $E$ and the vertex set $V$. We denote a weight matrix as $W$, wherein $W = (W_{ij}, \forall i, j \in V)$. For the vertex set $V$, let $D$ denote the diagonal matrix whose diagonal entries are row sums of $W$. For a fixed policy, a MDP is a graph with vertices being the states and edges representing transitions between states. Hence, we can construct the matrices $W, D$ for MDPs. In case of a discrete state MDP, the adjacency matrix $A = (A_{ij}, \forall i, j \in V)$ is used as the weight matrix. $A_{ij} = 1$ when states $i$ and $j$ are connected, and $A_{ij} = 0$ when $i$ and $j$ are not connected. For a continuous state MDP, $V$ is a subset of the states. In such a scenario the weight matrix is meaningfully represented by the Gaussian kernel matrix. Given a set $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ of $n$ data points in $V$, the $n \times n$ Gaussian kernel matrix $K = (K(x_i, x_j))$ is given by

$$K(x_i, x_j) = \exp\left(-\frac{||x_i-x_j||}{2\sigma^2}\right),$$

(2)

where $\sigma > 0$ is a positive scaling constant. Note that $K$ is a similarity matrix which assigns the nearby states a higher value, a fact evident from (2).

For a given graph $G$, the combinatorial Laplacian operator $L$ is defined as $L = D - W$, where $W = A$ if MDP has discrete state space. Similarly, the normalized Laplacian operator $N_L$ is defined as $N_L = D^{-1/2}LD^{-1/2}$. An important result in [18] is that by carrying out the spectral analysis of the operator $N_L$ we can use its eigen vectors

\[ \text{(2)} \]
which is affected by the underlying reward structure. Let \( W \) prevents the agent from choosing unfavourable states. The weight matrix so defined captures the nearness of states of goal oriented tasks, the immediate rewards are 0 we also get to observe the immediate rewards which can be used to capture preferred connectivity. While in the case prefer states that are not mine states (which yield high negative rewards).

For non-goal states, it might not hold true for MDPs with a general reward structure. In any case, the agent’s decision can be influenced by shaping the immediate rewards. If used, the shaped rewards must preserve the optimal policy structure which can be guaranteed if the reward shaping functions satisfy certain conditions and are potential functions (see [23]). We denote the shaped reward for state transition from a state \( s \) to \( s' \) as \( R(s, s') \). Using the shaped rewards \( \tilde{R} \), we define the weight matrix \( W_R \) in a manner which prevents the agent from choosing unfavourable states. The weight matrix so defined captures the nearness of states which is affected by the underlying reward structure. Let \( W_R = (w_{r}(s, s'), s \in S, s' \in S) \), where

\[
w_{r}(s, s') = \frac{\exp^{\beta R(s')}}{\sum_{s'' \in S} \exp^{\beta R(s'')}}. \quad (3)
\]

In (3), \( \beta > 0 \) is a positive constant that models affinity. The parameter \( \beta \) allows one to incorporate aversion (\( \beta < 0 \)) or preference (\( \beta > 0 \)) to risk (see [12]). Risk sensitivity is used to avoid unfavourable states. An unfavourable state \( s' \) can have high negative reward \( \tilde{R}(s') \), in which case \( W_r(s, s') \approx 0 \). This way an undesirable state has negligible weight. Using the weight matrix \( W_R \), we compute the matrix \( D \), find the combinatorial and normalized Laplacian operators and the diffusion matrix.

### 3.2 Basis Construction

The basis functions for the state-action value function are selected by analysing the spectra of the random walk diffusion matrix \( D_R \). We choose the top \( m \) eigen values of \( D_R \) and the corresponding eigen vectors to form the matrix \( \Psi \). The \( i^{th} \) row of the matrix \( \Psi_{|V| \times m} \), i.e., \( \psi(i) \), \( i \in 1, 2, \ldots |V| \), is the feature vector of state \( s_i \) that is in the vertex set \( V \) or in the sample set (w.r.t continuous state space MDP). Hence, \( \psi(s_i) = \psi(i) \). In order to approximate the action-value functions of MDPs we repeat the above basis functions for each of the actions as follows:

\[
\phi(s_i, a) = [I(a, a_1)\psi(s_i), \ldots, I(a, a_{|A|})\psi(s_i)]. \quad (4)
\]

In (4), \( \phi(s_i, a) \) is the feature vector corresponding to the state-action pair \( (s_i, a) \). \( I(a, a_j) \) is the indicator function, that is if \( a = a_j \), then \( I(a, a_j) = 1 \) else \( I(a, a_j) = 0 \). Note that since \( \psi(s_i) \in \mathbb{R}^m, \phi(s_i, a) \in \mathbb{R}^k \), where \( k = m \times |A| \). The \( j^{th} \) column of \( \Phi \) matrix gives the \( j^{th} \) basis function value for all state-action pairs. Thus the action-value function \( Q^\pi(s, a) \) for a policy \( \pi \) is approximated as:

\[
Q^\pi(s, a) = \sum_{i=1}^{k} \phi_i(s, a)w^\pi(i), \quad (5)
\]
where \( w^\pi \in \mathbb{R}^k \) is the weight vector.

For continuous state space MDPs, it is not possible to obtain trajectory samples for all state-action pairs. Thus the basis constructed for the explored state-action pairs is used for extending the basis vectors to unexplored state-action pairs. This extension is possible by the Nyström interpolation method [11]. Let \( \lambda_j \) be an eigenvalue of the normalized Laplacian \( N_L \). Then \( 1 - \lambda_j \) is the corresponding eigenvalue of the diffusion matrix \( D_R \). For an unexplored state \( \tilde{s} \), the feature \( j \) can be computed using the Nyström extension as follows:

\[
\psi_j(\tilde{s}) = \frac{1}{1 - \lambda_j} \sum_{s=1}^{\lfloor V \rfloor} \frac{K(\tilde{s}, s)}{\sqrt{d(\tilde{s})d(s)}} \psi_j(s),
\]

(6)

where \( d(s) = \sum_{z=1}^{\lfloor V \rfloor} K(s, z) \) and \( K(s, z) \) is as given by (2).

### 3.3 Shaping-based RPI Algorithm

The SRPI algorithm (see Algorithm 1) is based on the RPI algorithm. The SRPI algorithm collects a set of trajectory samples using a random policy. Based on the samples, the weight matrix \( W_R \) is constructed and eigen values of the diffusion matrix \( D_R \) are obtained. State-action features \( \phi(s, a), \forall s \in S, a \in A(s) \) are constructed using (4). The approximate value function is evaluated using the LSTDQ algorithm [17] (a variant of LSTD algorithm, see Algorithm 2) and improved using the LSPI algorithm. Since model information is not available, the LSTDQ algorithm learns it from the sample trajectories. The integer \( t \) is chosen large enough to ensure convergence.

### Algorithm 1 SRPI

1. Sample collection: Collect a set of samples \( D_S \) using a random policy \( \pi_0 \)
2. Diffusion matrix construction:
   1. Construct the weight matrix \( W_R \) using shaped rewards \( \tilde{R} \)
   2. Compute the random walk diffusion matrix \( D_R \)
3. Feature Matrix construction:
   1. Generate the eigen vectors of \( D_R \)
   2. Construct the matrix \( \Psi \) by using the top \( m \) eigen vectors of \( D_R \)
   3. Construct \( \Phi \) using (4)
4. for \( i = 0, 1, 2, \ldots, t - 1 \) do
5. Policy Evaluation Step: \( w^{\pi_i} = LSTDQ(D_S, \pi_i, k) \)
   \( \tilde{Q}^{\pi_i} = \Phi w^{\pi_i} \)
6. Policy Improvement Step:
   Set \( \pi_{i+1}(s) = \arg \max_a \tilde{Q}^{\pi_i}(s, a), \forall s \in S \)
7. end for
8. Return \( \pi_t \)

### 4 SIMULATION AND EXPERIMENTAL STUDIES

In this section, the performance of the reward shaping based graph Laplacian framework for value function approximation is evaluated and compared with RPI algorithm on two benchmark domains - grid world and mountain car. We also compare our work with the CRPI algorithm [32] on the mountain car task. Additionally in our experiments we demonstrate the usefulness of SRPI algorithm on the foraging problem [21, 26], which is a well studied problem in ecology.
Algorithm 2 LSTDQ ($\mathcal{D}_S, \alpha, \pi, k$)

1. $\tilde{A} \leftarrow 0$
2. $\tilde{b} \leftarrow 0$
   for $(s, a, r_a(s), s') \in \mathcal{D}_S$ do
3. \quad $\tilde{A} \leftarrow \tilde{A} + \phi(s, a)(\phi(s, a) - \alpha\phi(s', \pi(s')))\top$
4. \quad $\tilde{b} \leftarrow \tilde{b} + \phi(s, a)r_a(s)$
   end for
6. $w^\pi \leftarrow \tilde{A}^{-1}\tilde{b}$
8. Return $w^\pi$

Figure 3: Three Room Task. Here the agent needs to start from S and reach the goal-state G. Each room is of size $20 \times 21$ and the walls that separate the room cause discontinuities and make the representation based on primitive functions ineffective.

4.1 Three Room Problem

The three room environment consists of three adjacent rooms each of size $20 \times 21$. The walls that separate the rooms have doors to allow access from one room to the next (see Fig. 3). The agent needs to learn the quickest possible way from the start state (labelled S in the figure) to the goal state (labelled G in the figure). The agent is only given a reward of 10 on reaching the goal state. The optimal value function is shown in Fig. 4. In [19], the above problem was introduced to show the performance of PVFs.

In our analysis, we initially used the Gaussian kernel matrix $K$ as the weight matrix for constructing the diffusion matrix $D_R$. In order to construct the kernel matrix, we evaluated the optimal value function $J^*$ for the three room problem and used it in the kernel matrix generation as follows:

$$K(x_i, x_j) = \exp\left(-\frac{|J^*(x_i) - J^*(x_j)|}{2\sigma^2}\right), \quad (7)$$

where $\sigma = 0.1$. The first eigen function of $D_R$ constructed using $K$ is as shown in Fig. 4. We observe that the first eigen vector of the kernel matrix in (7) is a close approximation to the optimal value function. This shows that any similarity matrix other than the adjacency matrix can also be used to generate the basis functions.

We further analyse the performance of PVFs and RPVFs in this task. In our experiments, we shape the reward structure of the three room task. Specifically while constructing the diffusion matrix $W_R$, we treat the three room grid as a $21 \times 20$ grid, which has high negative rewards in the place of the wall. This is a different approach compared to [19]. The PVFs and RPVFs generated for the three room problem are shown in Fig. 5. It is observed that the eigen functions generated in both cases are similar. This shows that the PVFs of the three room problem can be recovered even when the walls are absent if we assign appropriate negative rewards for those cells corresponding to the “wall” (inaccessible) states.

4.2 Variants of Grid World

In this section, we consider another instance of goal-based MDP (see Table 1). The goal-based MDP is a $N \times N$ grid. The state space for an $N \times N$ grid is given by $S = \{(x, y), x = 1, \ldots, N, y = 1, \ldots, N\}$, where $(1, 1)$ denotes the bottom-left cell and $(N, N)$ the top-right cell. The allowable actions are to move up, down, right or left. $(N, N)$ is the goal-state. The agent receives a reward of 10 on reaching the goal state and actions in the intermediate states do
Figure 4: On the left is the optimal value function and on the right is the first eigen function of the matrix $K$ generated using $J^*$ and $\sigma = 0.1$.

Figure 5: First two on the left are the eigen functions of the $W$ matrix and two on the right are the eigen functions of the $WR(\beta = 0.1)$ matrix.

not receive any reward. It is evident that the learning process can be sped up if the agent is rewarded for those actions that take it either up or right.

We consider a grid for $N = 5$. The optimal value function is shown in the top most plot of Fig. 6. We chose $k = 4$, i.e., 4 eigen functions corresponding to 4 largest eigen values of the diffusion matrix $D_R$ constructed from the adjacency matrix $A$. We ran the RPI algorithm [18] with the diffusion matrix constructed from $A$ and the result is shown (top-right) in Fig. 6. Notice that the value function learnt by the RPI algorithm does not quite resemble the profile of the optimal value function and consequently resulted only in a moderately good policy. We evaluated the policy $\pi_W$ returned by RPI and it turned out that $\sum_{s \in S} J_{\pi_W}(s) = 1132$ as opposed to $\sum_{s \in S} J^*(s) = 1887$. Further, we also ran the RPI algorithm by retaining the diffusion matrix, but provided additional reward shaping feedback using the shaping function $\tilde{R}$ during the learning phase (see Table 1). Even in this case the profile of the learnt value function

Table 1: On the left is the grid world task with a reward of 10 in the goal-state G. On the right is the potential reward shaping function.
Figure 6: Shows the optimal value function (left, top row) and the learned value functions (other three) for the grid world domain in Table 1. Notice that learning using the RPVF (right, bottom row) is better than the learning using PVF with/without reward shaping.

Figure 7: Comparing the profiles of eigen functions of $W$ and $W_R$. The first two eigen functions of $W$ and $W_R$ were identical and hence not presented.

did not change, and the resulting policy performed only moderately. The SRPI algorithm was run for this experiment by using the shaped rewards for constructing the weight matrix $W_R$. In this case, the profile of the learnt value function resembles the optimal value function. Further, we also observed that the policy $\pi_{W_R}$ returned by SRPI in this case performed better than $\pi_W$ (with/without reward shaping feedback), i.e., $\sum_{s \in S} J_{\pi_{W_R}}(s) = 1660$.

The reason why the RPVFs perform better than the PVFs can be explained by looking at the corresponding eigen functions (see Fig. 7). The PVFs and RPVFs corresponding to the first two largest eigen values were the same. However, the third and fourth eigen functions differed (see Fig. 7). We can see from (Figures 6 and 7), that this difference in eigen function shows up in the difference in the profiles of the corresponding learnt value functions.

We also compared the performance of PVFs and RPVFs in a variant of the grid world problem, where, in addition to the goal-state, there are certain mine states with negative rewards (see Table 2). We chose these mine states at random and then compared the performances across 10 such different random grid world problems and for each problem we averaged the result across 10 initial policies. We observed that in 9 out of the 10 systems, SRPI (generated for $\beta = 0.1$) significantly outperforms the policy learnt using the RPI. These results are shown in Fig. 8.

4.3 Mountain Car Problem
Table 2: On the left is the grid world task with mines (marked as X). On visiting a mine state the agent receives a random reward between $-1$ to $-5$. On the right is the optimal value function for the mine task.

A classical benchmark problem in continuous state space is the mountain car problem. The problem deals with an underpowered car that has to be driven up a hill. The continuous state space $s_t, t \geq 0$ of the system comprises of position $p_t$ and velocity $v_t$ of the car. In every state configuration, three actions are possible. The agent can choose to have full acceleration forward (denoted $+1$), full acceleration backward (denoted $-1$) or not apply any force (denoted $0$) on the car. Based on the action, the position $p_t$ and velocity $v_t$ are updated as follows:

$$p_{t+1} = \text{bound}(p_t + v_t)$$

$$v_{t+1} = \text{bound}(v_t + 0.001a_t - 0.0025 \cos(3p_t))$$

where the bound operation ensures that $p_t \in [-1.2, 0.6]$ and $v_t \in [-0.07, 0.07]$, $\forall t$. An episode ends successfully if the car reaches the goal state (i.e., when $p_t \geq 0.5$) or terminates unsuccessfully if the car has not reached the goal state even after 500 steps. The initial state of the car is $p_0 = -0.6, v_0 = 0$. The discount factor is set to 0.99.

Generating PVFs and RPVFs in continuous domain such as mountain car, poses significant challenge. The problem is handled by generating the weight matrix from the data stored using off-policy sampling. The basis function for the novel states that are encountered later during learning are estimated from the previously calculated basis function using Nystrom Extension method (see Section 3.2). In practice for faster evaluation, the evaluation of the basis functions is done only for 10% of the sample states and for novel states encountered each basis function is estimated by averaging the basis functions of the nearest neighbour states. We employ the k-nearest neighbour search algorithm to find the nearest neighbour based on the Euclidean distance between states.

Another challenge in continuous MDP is to select a subset of the sampled data to compute basis functions for the states. A random policy can be used for this purpose. A small set of points is sufficient to form the sample set, which can be further used in the learning phase. Random sampling can be done where a number of data points are chosen randomly to form the weight matrix. To store important samples such as of state with high reward or goal state, trajectory based sampling is used. Let $X = \{x_0, x_1, x_2 \ldots \}$ be the sampled points. We follow the sequence of steps below:

1. Start with an empty set $S$ and $\epsilon > 0$
2. $S := \{x_0\}$ and $X := X \setminus \{x_0\}$
3. For all the sampled points $x_i \in X$, and $\forall x_j \in S$, if $\text{distance}(x_i, x_j) > \epsilon$, then $S := \{S, x_i\}$ and $X = X \setminus x_i$

We get set $S$ of data points for which no two data points are $\epsilon$ close to each other. Suppose a state $x_j$ is one of the $k$ nearest neighbours of $x_i$, where $k > 0$. Then, using such data points we create a graph where an edge is inserted between two such nearest neighbour states $x_i$ and $x_j$. For our case we have chosen $k = 20$. 

11
In the reward extraction stage (see Fig. 2), the shaped reward is $C \times |v_t|^2$, where $C$ is a constant. This is done to favour states with higher kinetic energy. The performance of RPVFs and PVFs are shown in Figures 10, 11. The plots show the number of steps taken to reach the goal state. Experimentally we find that the above reward shaping function is extremely useful and it outperforms PVFs.

In addition to the subsampling techniques explained earlier, $k$-means clustering technique can also be used to sample data points. This has been proposed in [32]. Combining with the previously mentioned reward shaping, RPVFs and PVFs can be generated. The performance of PVFs and RPVFs with this subsampling technique is shown in Fig. 11. The number of clusters chosen for subsampling is 500. The SRPI is seen to outperform CRPI in Fig. 11.

4.4 Foraging Problem

Foraging is the process of exploration of food sources by animals. It includes the act of searching for food and also avoiding predation while looking for food and reproduction. While foraging, every animal is subjected to natural conditions and multiple decisions concerning which of multiple actions to perform as well as how best to perform on the undertaken activity. For instance, when an animal is searching for food across a vast stretch of land, it must take into account the natural landscapes in the path (mountains, valleys, desert etc), availability of nutritious food (fruits, insects etc), chance of survival from predation and its own level of fitness. In all these situations the animal...
Figure 9: Mountain Car

Figure 10: The performance of RPI and SRPI algorithms for $k = 20$ and $k = 40$ basis functions

Figure 11: The performance of RPI, CRPI and SRPI algorithms for $k = 30$ basis functions
must decide whether to go for nutritious food which is far-off (low chance of predation survival or fitness is high) or be contented with not-so nutritious food (to avoid predation or because fitness is low). The scenario renders itself to sequential decision-making and can be modeled as a MDP with appropriate state and action variables along with a cost structure. However it should be noted that the foraging decisions are dependent on the environment which is nonstationary due to the depletion of food sources and changing weather conditions. This nonstationarity make it hard to solve the foraging problem.

Figure 12: Terrain 1

Suppose a forager can forage in one of \( N = n_1 \times n_2 \) distinct patches on each day. Each patch is characterized by energy cost (energy exhausted on the way to the patch), predation chances, and energy value of food items. These costs can be encapsulated as a single cost variable associated with a patch. The nutrition content of the food can be captured in a reward vector.

The primary objective of the forager is to increase its nutritional intake and the secondary objective is to keep away from costly terrains. While the forager moves in a way to maximize its primary objective, its local decisions are dictated by the secondary objective. These two conflicting objectives can be captured by constructing RPVF based on the secondary objective and use them to learn the value function based on the primary objective.

Figure 12 shows Terrain 1. The reward for the fruits in the tree and that of the thick grass is 10, elsewhere the reward is 2. The reward for the water body is 5. The cost of climbing the mountain is 20 and swimming 10. We ran the SRPI and RPI for the foraging problem in this terrain. The policy learnt by RPI yielded a reward of 495 while incurring a cost of 1112. However, the policy learnt by SRPI yielded a reward of 495 while incurring a cost of 200. We also ran the foraging experiment on Terrain 2 which is similar to Terrain 1 but with larger area. Even in this case, SRPI learnt a policy with reward 1180 and cost 110, while the policy learnt by RPI yeilded only a reward of 945 but at the same time incurred a cost of 204. The results on the two terrains can be qualitatively summarized as follows:

- In both terrains SRPI outperformed the RPI.
- The smaller terrain (Terrain 1) was difficult since it was harder to avoid the obstacles that were closer to each other in comparison to Terrain 2 wherein the obstacles were far apart.

5 CONCLUSION

This paper presents a new reward transformation based graph Laplacian framework for basis representation and adaptation in RL. The framework we propose is helpful in finding a compact representation for the value function. For
the linear architecture of the approximate value functions, basis functions are computed using the SRPI method. The SRPI algorithm mimics the classic policy iteration scheme [25]. The performance of SRPI is compared with CRPI and RPI for the mountain car problem and with RPI for the grid world and foraging problem. It is shown that the reward shaping based Laplacian captures the neighbourhood information well when compared to PVFs. The experimental results show that the SRPI algorithm requires fewer number of steps on an average to compute the optimal set of basis functions. It is also observed that similarity matrices other than the diffusion matrix can be used to generate the features. Reward transformation is shown to aid learning, and the SRPI algorithm is seen to outperform the policy learnt using PVFs. As a future enhancement of this work, we will look at a rigorous analysis of the SRPI algorithm and apply it to problems in energy harvesting (EH) sensor networks and traffic control.

References


