Representation Learning on Graphs: A Reinforcement Learning Application

Sephora Madjiheurem
University College London, UK
sephora.madjiheurem.17@ucl.ac.uk

Laura Toni
University College London, UK
l.toni@ucl.ac.uk

Abstract

In this work, we study value function approximation in reinforcement learning (RL) problems with high dimensional state or action spaces via a generalized version of representation policy iteration (RPI). We consider the limitations of proto-value functions (PVFs) at accurately approximating the value function in low dimensions and we highlight the importance of features learning for an improved low-dimensional value function approximation. Then, we adopt different representation learning algorithms on graphs to learn the basis functions that best represent the value function. We empirically show that *node2vec*, an algorithm for scalable feature learning in networks, and *Variational Graph Auto-Encoder* constantly outperform the commonly used smooth proto-value functions in low-dimensional feature space.

1 INTRODUCTION

In reinforcement learning (RL), an agent, or decision maker, takes sequential actions and observes the consequent rewards and states, which are unknown *a priori*. These sequent observations improve the agent's knowledge of the environment with the final goal of learning the optimal policy that maximizes the long term reward. The learning control problem is usually formulated as Markov decision process (MDP), where each state has an associated value function, which estimates the expected long term reward under some policy (usually the optimal one). Classical MDP solutions represent the value function by

a lookup table, with one entry for each state. However, this does not scale with the state (and implicitly also action) space dimension, leading to slow learning processes in high-dimensional reinforcement learning problems. Approximated reinforcement learning addresses this problem by learning a function to properly approximate the true value function. In the literature, many types of functions have been studied [Kaelbling et al., 1996, Sutton and Barto, 1998].

In this work, we study linear value function approximation, where the value function is represented as a weighted linear sum of a set of features (called *basis function*). Under this framework, one of the main challenges is to identify the right set of basis functions. Typical linear approximation architectures such as polynomial basis functions (where each basis function is a polynomial term) and radial basis functions (where each basis function is a Gaussian with fixed mean and variance) have been studied in the case of reinforcement learning [Lagoudakis, 2003]. These architectures make the assumption that the underlying state space has Euclidiean geometry. However, in realistic scenarios, the MDP's state space is likely to exhibit irregularities, as in the example of Figure 1. In Figure 1(b), neighboring states can have values that are far apart (such as states on opposite sides of a wall). In such cases, these traditional parametric functions may not be able to accurately approximate value functions.

Consequently, other basis functions have been studied to address this issue. Example of such methods include Fourier basis [Konidaris et al., 2011], diffusion wavelets, [Mahadevan and Maggioni, 2006], Krylov basis [Petrik et al., 2010] and Bellman Error Basis Function [Parr et al., 2007, Parr et al., 2008]. In particular, work by [Mahadevan and Maggioni, 2007] introduces the *representation policy iteration* (RPI), a spectral graph framework for solving Markov decision processes by jointly learning representations and optimal policies in irregular state spaces. In their work, the authors first note that an MDP can be intuitively
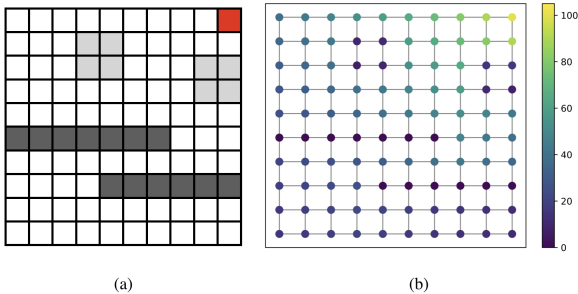
Figure 1: (a) Maze environment. The dark grey squares are strict walls, while the light grey square are difficult access rooms. The red square is the goal room. (b) Optimal value function computed using value iteration [Montague, 1999].

represented using a *state-graph*, in which the states are the nodes and the transition probability is the similarity matrix. Then, under the assumption that the value function is usually modelled as a diffusion process over the state-graph (and therefore it is a smooth function), they approximate the value function (smooth signal on the graph) as a linear combination of the first Laplacian eigenmaps on the state-graph. A small set of these features, known as *proto-value functions* (PVFs), reliably approximate the value function under the smoothness prior. However, in this paper, we argue that the value function is not necessarily a smooth function on the state-graph representing the MDP. Hence, defining the basis functions under a smoothness prior might be suboptimal. Therefore, there is a need to automatically learn compact basis functions that capture the geometry of the underlying state space. Given the success of recent node embedding models [Grover and Leskovec, 2016, Kipf and Welling, 2016b, Ribeiro et al., 2017, Donnat et al., 2018], we propose to investigate representation learning on graphs algorithms to learn basis functions in the linear value function approximation.

The idea behind recent successful representation learning approaches is to learn a mapping that embeds the nodes of a graph as low-dimensional vectors. They aim to optimize the representations so that geometric relationships in the embedding space preserve the structure of the original graph. [Hamilton et al., 2017] surveys recent representation learning on graph methods. Therefore, in this work, we generalize the RPI algorithm [Mahadevan and Maggioni, 2007] to allow different basis functions, and analyze the performance of several representation learning methods for value function approximation.

The rest of this paper is structured as follows: in Section 2 we introduce background material, providing

details on Markov decision processes, value function approximation, and the representation learning algorithm used in this work. The General Representation Policy Iteration algorithm is described in Section 3. In Section 4, we discuss experimental results and proceed to summarize the main findings and give direction for future work. We finally conclude in Section 5.

2 BACKGROUND

2.1 Markov decision process (MDP)

Markov decision processes are discrete time stochastic control processes that provide a widely-used mathematical framework for modeling decision making strategies under uncertainty. Specifically, at each time step, the process is in some state s, and the agent can choose any action a that is available in state s. As a consequence of the action taken, the agent finds itself in a new state s' and observes an instantaneous reward r. We define a discrete MDP by the tuple $M = (S, A, P, R)$, where S is a finite set of discrete states, A a finite set of actions, P describes the transition model − with $P(s, a, s')$ giving the probability of moving from state s to s' given action a − and R describes the reward function − with $R(s, a)$ giving the immediate reward from taking action a in sate s. Given a policy $\pi : S \mapsto A$, a value function V^π is a mapping $S \mapsto \mathbb{R}$ that describes the expected long-term discounted sum of rewards observed by the agent in any given state s when following policy π. Solving an MDP requires to find a policy that defines the optimal value function V^*, which satisfies the following constraints:

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

This recursive equation is known as the *standard form* of Bellman's equation. The optimal policy is a unique solution to the Bellman's equation and can be found by dynamic programming, iteratively evaluating the value functions for all states.

2.2 Value Function Approximation

In large state spaces, computing exact value functions can be computationally intractable. A possible solution is to estimate the value function with function approximation (value function approximation method) [Bertsekas and Tsitsiklis, 1996]. Commonly, the value function is approximated as a weighted sum of basis functions [Montague, 1999, Mahadevan, 2007, Konidaris et al., 2011, Lagoudakis, 2003]:

$$\phi_1, \phi_2, \ldots, \phi_d : \quad \tilde{V}(s|\theta) = \sum_{i=1}^{d} \theta_i \phi_i(s) \approx V(s),$$

where d is the dimension of the features space.

The basis functions ϕ_i can be hand-crafted [Sutton and Barto, 1998] or automatically constructed [Mahadevan and Maggioni, 2007], and the model parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$ are typically learned via standard parameter estimation methods such as least-square policy iteration (LSPI) [Lagoudakis, 2003]. However, how to properly design the set of basis for a data-efficient function approximation framework is still an open question. The main question is how to find the set of basis ϕ that is *low-dimensional* (to ensure a data-efficient learning) and yet a *meaningful representation* of the MPD (to reduce the suboptimality due to the approximation of the value function).

The representation policy iteration algorithm (RPI) was introduced in [Mahadevan and Maggioni, 2007] to address this problem. It is a three steps algorithm consisting of (1) a sample collection phase, (2) a representation learning phase and (3) a parameter estimation phase. RPI is described in further details in Section 3. In this work, we propose to generalize RPI to allow different representation learning methods. In particular, we first observe that state space topologies of MDPs can be intuitively modeled as (un)directed weighted graphs, with the nodes being the states and the transition probability matrix being the similarity matrix. When the transition probabilities are unknown, we can construct a graph from collected samples by connecting temporally consecutive states with a unit cost edge. Therefore, similarly to [Mahadevan, 2007], we propose to construct the graph from collected samples of an agent acting in the environment given by the MDP. We then learn representations on the graph induced by the MDP using node embedding methods. Finally, we use the learned representations to linearly approximate the value function. In the next section, we describe the node embedding models that we exploit within this framework.

2.3 Representation Learning on Graph

We propose to use the following learned node embedding models as basis functions for the value function approximation.

Node2Vec *Node2vec* [Grover and Leskovec, 2016] is an algorithmic framework for learning continuous feature representations for nodes in networks. It is inspired by the powerful language model *Skip-gram* [Mikolov et al., 2013] which is based on the hypothesis that words that appear in the same context share semantic meaning. In networks, the same hypothesis can be made for nodes, where the context of a node is derived by considering the nodes that appear in the

same random walk on the graph. Therefore, node2vec learns the node embeddings based on random walk statistics. The key is to optimize the node embeddings so that nodes have similar embeddings if they tend to co-occur on short (biased) random walks over the graph. Moreover, it allows for a flexible definition of random walks by introducing parameters that allow to interpolate between walks that are more breadth-first search or depth-first search.

Specifically, for a graph $G = (\mathcal{V}, \mathcal{E}, W)$ (where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ a set of edges and W the weight matrix) and a set $\mathcal{W}$ of T biased random walks collected under a specific sampling strategy on the graph G, node2vec seeks to maximize the log-probability of observing the network neighborhood of each node $u \in \mathcal{V}$ conditioned on its features representations, given by f (a matrix of size $|V| \times d$ parameters, where d is the dimension of the feature space):

$$\max_f \sum_{w \in \mathcal{W}} \sum_{t=1}^{T} \log Pr(N_w(u_i)|f(u_i)),$$

where $N_w(u_i)$ describes the neighborhood of the ith node in the walk w.

Struc2Vec By introducing a bias in the sampling strategy, node2vec allows to learn representations that do not only focus on optimizing node embeddings so that nearby nodes in the graph have similar embeddings, but also consider representations that capture the structural roles of the nodes, independently of their global location on the graph. The recent node embedding approach, *struc2vec*, proposed by [Ribeiro et al., 2017] addresses the problem of specifically embedding nodes such that their structural roles are preserved. The model generates a series of weighted auxiliary graphs G_k (with $k = 1, 2, ...$) from the original graph G, where the auxiliary graph G_k captures structural similarities between nodes k-hop neighborhoods. Formally, let $R_k(u_i)$ denote the ordered sequence of degrees of the nodes that are exactly k-hops away from u_i, the edge-weights, $w_k(u_i, v_j)$, in the auxiliary graph G_k are recursively represented by the structural distance between nodes u_i and v_j defined as

$$w_k(u_i, v_j) = w_{k-1}(u_i, v_j) + d(R_k(u_i), R_k(u_j)),$$

where $w_0(u_i, v_j) = 0$ and $d(R_k(u_i), R_k(u_j))$ is the distance between the ordered degree sequences $R_k(u_i)$ and $R_k(u_j)$ computed via dynamic time warping [Ribeiro et al., 2017].

Once the weighted auxiliary graphs G_k are computed, struc2vec runs biased random walks over them and proceeds as node2vec, optimising the log-probability

of observing a network neighborhood based on these random walks.

GraphWave The *GraphWave* algorithm as proposed by [Donnat et al., 2018] takes a different approach to learning structural node embeddings. It learns node representations based on the diffusion of a spectral graph wavelet centered at each node. For a graph G, $\mathcal{L} = D - A$ denotes the graph Laplacian, where A is the adjacency matrix and D is a diagonal matrix, whose entries are row sums of the adjacency matrix. Let U denote the eigenvector decomposition of the graph Laplacian $\mathcal{L} = U \Lambda U^T$ and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{|V|})$ denote the eigenvalues of $\mathcal{L}$. Given a heat kernel $g_s(\lambda) = e^{-s\lambda}$ for a given scale s, GraphWave uses U and g_s to compute a vector $\boldsymbol{\psi}_u$ representing diffusion patterns for node u as follows:

$$\boldsymbol{\psi}_u = U \mathrm{diag}(g_s(\lambda_1), g_s(\lambda_2), \ldots, g_s(\lambda)_{|V|}) U^T \delta_u$$

where δ_u is the one-hot indicator vector for node u. Then, the characteristic function for each node's coefficients $\boldsymbol{\psi}_u$ is computed as

$$\phi_u(t) = \frac{1}{|V|} \sum_{m=1}^{|V|} e^{it\Psi_{mu}}$$

Finally, the structural node embedding $f(u)$ for node u is obtained by sampling the paramatric function $\phi_u(t)$ at d evenly spaced points $t_1, \ldots, t_d$:

$$f(u) = \big[\mathrm{Re}(\phi_u(t_i)), \mathrm{Im}(\phi_u(t_i)) \big] t_1, \ldots, t_d.$$

Variational Graph Auto-Encoder An alternative is to adopt *Variational Graph Auto-Encoder* (VGAE) to learn graph features. The Variational Graph Auto-Encoder proposed by [Kipf and Welling, 2016b] is a latent variable model for graph-structure data capable of learning interpretable latent representations for undirected graphs. Lets consider an undirected and unweighted graph $G = (\mathcal{V}, \mathcal{E}, A)$ with $N = |\mathcal{V}|$ nodes and feature matrix $X \in \mathbb{R}^{N \times F}$ with the ith row representing the feature of node i. The VGAE inference model is given by

$$q(Z|X, A) = \prod_{i=1}^{N} q(z_i|X, A),$$

with $q(z_i|X, A) = \mathcal{N}(z_i|\mu_i, \mathrm{diag}(\sigma_i^2))$, wheare μ is the matrix of mean vectors and σ is the variance vector given respectively by $\mu = \mathrm{GCN}_\mu(X, A)$ and $\log \sigma = \mathrm{GCN}_\sigma(X, A)$. The parameters to the Graph Convolutional Neural Networks (GCNs) are denoted by θ_i. The matrix $Z \in \mathbb{R}^{N \times D}$ is the matrix of latent

variables whose rows are the vectors z_i. The generative model is given by

$$p(A|Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij}|z_i.z_j)$$

with $p(A_{ij} = 1|z_i, z_j) = \sigma(z_i^T z_j)$, where $\sigma(\cdot)$ denotes the sigmoid function.

The learning phase optimises the following loss function w.r.t to the variational parameters θ_i:

$$L = \mathbb{E}_{q(Z|X,A)}[\log p(A|Z)] - \mathrm{KL}[q(Z|X, A)||p(Z)],$$

where $\mathrm{KL}[q(\cdot)||p(\cdot)]$ denotes the Kullback-Leibler divergence between $q(\cdot)$ and $p(\cdot)$. An additional Gaussian prior $p(Z) = \prod_i p(z_i) = \prod_i \mathcal{N}(z_i|0, \boldsymbol{I})$ is taken.

3 GENERAL REPRESENTATION POLICY ITERATION

Within the context of approximated value function, the representation policy iteration algorithm (RPI) was introduced in [Mahadevan and Maggioni, 2007] to learn the approximating function. RPI is a three step algorithm consisting of (1) a sample collection phase, to build a training dataset with quadruples $\{(s_i, a_i, s_{i+1}, r_i)\}$; (2) a representation learning phase that defines a set of basis functions; and (3) a parameter estimation phase, in which the coefficients of the linear approximation are learned. A generalized version of the RPI algorithm [Mahadevan, 2007] is described in Algorithm 1.

In the original RPI, the representation learning phase is predefined. Namely, an undirected graph G is built from the available data set $\mathcal{D}$, and the normalized Laplacian is computed on graph G. The d-dimensional basis functions $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_d]$ are the d smoothest eigenvectors of the graph Laplacian and are known as *proto-value functions* (PVFs). The key is that given a state-graph that perfectly represents the MDP, the value function is modelled as a diffusion process over the graph (and therefore it is a smooth function). Under smoothness prior, the proto-value functions are a good choice of basis functions as they reliably represent smooth value functions.

In this paper, we argue that PVFs are optimal basis in ideal value function approximation problems, which provide an a priori knowledge of the graph. In practice, the graph is estimated from previous observations. Using the estimated graph as support to the value function does not necessarily preserve the smoothness assumption. To support this argument, we consider the environment depicted in Figure 1. To construct the estimated graph $\hat{G}$, we first collect samples by running

Algorithm 1 General Representation Policy Iteration

Input:
π_0: sampling strategy,
N: number of random walks to sample,
T: length of each walk,
d: dimension of the basis functions,
$embed()$: representation learning model,
ϵ: convergence condition for LSPI.
Output: ϵ-optimal policy π
1. Sample Collection Phase
Collect a data set $\mathcal{D}$ of T successive samples $\{(s_i, a_i, s_{i+1}, r_i), (s_{i+1}, a_{i+1}, s_{i+2}, r_{i+1}), \ldots\}$ by following sampling strategy π_0 for maximum T steps (terminating earlier if it results in an absorbing goal state).
2. Representation Learning Phase
Build basis function matrix $\phi = embed(\mathcal{D}, d)$.
3. Control Learning Phase
Using a parameter estimation algorithm such as LSPI or Q-learning, find an ϵ-optimal policy π that maximizes the action value function $Q^\pi = \phi\theta^\pi$ within the linear span of the basis ϕ.

	$\boldsymbol{v^T \mathcal{L} v}$
Estimated graph	14831.72
Weighted graph	5705.65

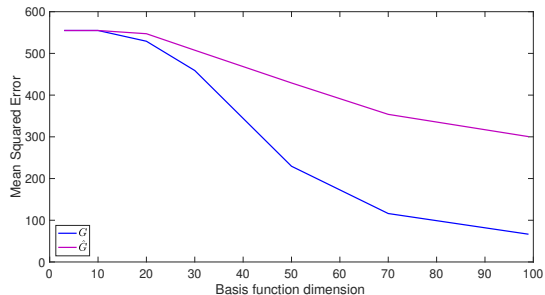Table 1: Analysis of the smoothness of the value function on different graphs.

Figure 2: MSE between the approximated value function and the true value function when using proto-value functions generated from two different graphs as basis functions. On the x axis we make the dimension of the basis function (the number of proto-value functions) vary. (Best seen in color.)

100 independent episodes starting at a random initial state and taking successive random actions until the goal state is reached (or after a timeout of $N_T = 100$ steps). We then connect temporally consecutive states with a unit cost edge. The ideal graph G is the graph with edges representing actual transition probabilities (i.e. edges between accessible states have weight 1, edges between an accessible state and a wall state have weight 0, and edges between an accessible or difficult access state and a difficult access state have weight 0.2). To measure the global smoothness of the value function on a graph, we use the following function:

$$\sum_{i,j \in \mathcal{E}} w_{ij}(v_i - v_j)^2 = \boldsymbol{v^T \mathcal{L} v}.$$

Where $\mathcal{L}$ is the graph Laplacian. In other words, if values v_i and v_j from a smooth function reside on two well connected nodes (i.e. w_{ij} is large), they are expected to have a small distance $(v_i - v_j)^2$, hence $\boldsymbol{v^T \mathcal{L} v}$ is small overall.

The actual smoothness for G and $\hat{G}$ is provided in Table 1. These results show a reduction of the smoothness when going from the ideal weighted graph to the estimated and unweighted graph (usually considered in realistic settings, when the transition probability is not known a priori).

As a consequence, the d smoothest proto-value functions of the estimated graph $\hat{G}$ do not allow a reliable reconstruction of the true value function. This

is due to the limited smoothness of the value function on $\hat{G}$. This phenomenon is verified in Figure 2. It shows the mean squared error (MSE) of the approximate value function for the environment of Figure 1. The MSE is computed in a least-square way using the true value function computed via value iteration [Montague, 1999].

To overcome this limitation, we propose to use the node embedding methods described in Section 2.3 to automatically learn the basis functions from the geometry of the underlying state space to further improve the performance. In the following, we describe how to apply these features learning methodologies within reinforcement learning strategies [1].

4 EXPERIMENTS

4.1 Set up

We consider the two-room environment (Figure 3(a)) used in [Mahadevan and Maggioni, 2007]. It consists of 100 states in total, divided into 57 accessible states and 43 inaccessible states representing walls. There is one goal state, marked in red and the agent is rewarded by +100 for reaching the goal state.

We also consider the obstacles-room environment depicted in Figure 3(b). In this environment, there are

---

[1] Code for reproducing the results is made available at https://github.com/LASP-UCL/Graph-RL
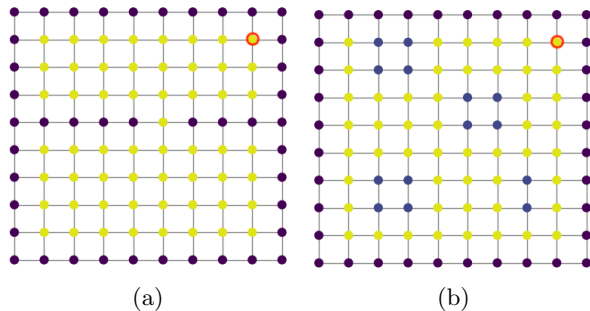
(a)   (b)

Figure 3: Two different maze environments. The purple nodes represent strict walls, while the blue nodes are difficult access rooms. All other nodes represent accessible rooms The node shown in red is the goal room. (Best seen in color.)

100 states in total, some of which are inaccessible (exterior walls) and 14 of which are accessible from neighbouring states with a fixed probability of 0.2 (moving obstacle or difficult access space). All the other states are reachable with probability 1.0. The agent is rewarded by +100 for reaching the state located at the upper-right corner.

We run and evaluate the General Representation Policy Iteration (GRPI) algorithm using embedding models from Section 2.3 to compute the basis functions in the second phase of the algorithm:

1. We first collect a set $\mathcal{D}$ of 100 sampled random walks, terminating when the goal state is reached, or after timeout $N_T = 100$. The sampling dynamic is as follows: starting from a random accessible sate, the agent takes one of the four possible actions (move up, down, left or right). If a movement is possible, it succeeds with probability 1.0. Otherwise, the agent remains in the same state. If the agent reaches the gold state, it receives a reward of +100, and is randomly reset to an accessible interior state for the next episode. We use off-policy sampling ($\pi_0 =$ random policy) to collect the samples, except in the case of node2vec, where the samples are generated under a biased random walk. We use grid search to find the optimal hyperparameters $p = 1$ and $q = 4$ that guide the walk according to [Grover and Leskovec, 2016].

2. We then use sample transitions in $\mathcal{D}$ to build an undirected graph where each state location is a node, and the transitions are represented by unit cost edges. We then run $embed(\mathcal{D}, d)$ with $embed \in \{$node2vec (n2v), struc2vec (s2v), variational graph auto-encoder (VGAE), GraphWave (GW)$\}$ for different choices of d. In the case of node2vec, we reuse the samples set to derive the
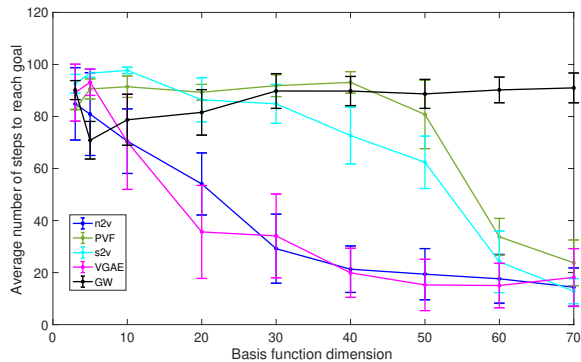
(a)
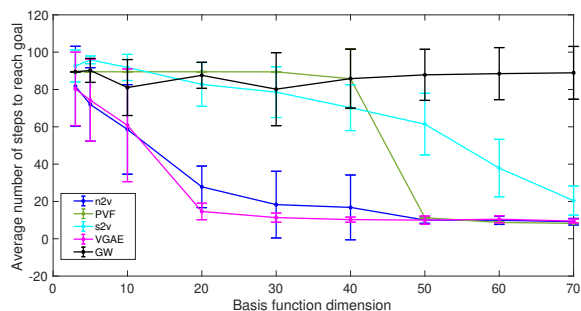
(b)

Figure 4: Average number of steps required to reach the goal steps using the various basis functions. On the x axis we make the dimension of the basis functions vary. Best seen in color. 4(a) corresponds to the two-room environment, while 4(b) corresponds to the obstacle-room environment.

node neighbourhoods used in the objective function.

3. We learned the parameters of the linear value approximation using the parameter estimation method LSPI [Lagoudakis, 2003] with the set of samples $\mathcal{D}$.

4. We used the policies learned by GRPI for each model to run simulations starting from each accessible states. We compare the performance of each models in terms of the average number of steps required to reach the goal. We also compare to the traditional PVF basis functions. The results for the two environments, averaged over 20 independent runs, are shown in Figures 4(a) and 4(b). Each run consists of episodes of a maximum of 100 steps, where each episode is terminated earlier if the agent reaches the goal state.
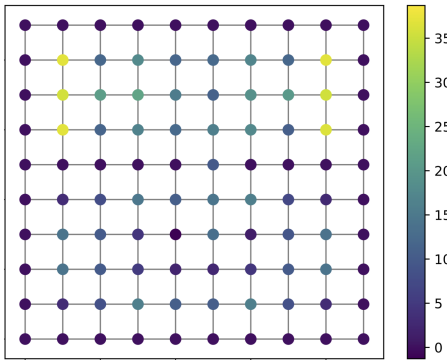
Figure 5: Approximate value function via GRPI using GraphWave basis function of dimension 70 on the two-room environment.

Figure 6: Average number of steps required to reach the goal steps using node2vec with varying parameters p and q. On the x axis we make the dimension of the basis functions vary.

4.2 Discussion

Figure 4(a) and Figure 4(b) show the average number of steps to reach the goal as a function of the dimension of the basis function. We first observe that the policy learned via the GraphWave basis function lead to very poor performances regardless of the dimension size. We investigate this phenomenon by looking at the approximate value function learned under these basis. The approximate state values are depicted in Figrue 5. Because GraphWave aims at learning embeddings that are exclusively structural, we hypothesise that they fail at capturing global network properties. In fact, the embeddings learned by GraphWave for the corner states in the two-room environment are equals, making it obviously impossible to learn different state values with linear approximation. This suggests that although the GraphWave is a powerful model for capturing structural information in networks, it is not a good choice of basis function for approximating value function on a graph.

On the other hand, we notice that although struc2vec was also designed to capture structural similarities between nodes, it also preserves the local properties of the graph by considering neighborhoods of different sizes [Ribeiro et al., 2017]. Hence, struc2vec is able to accurately approximate the value function even in graphs that have symmetrical structure such as the two-room environment.

Finally, the result show that VGAE and node2vec are good choices of basis functions for approximating the value function with low dimensional features. Namely, they both lead to good performances in terms of number of steps to reach the goal states with basis functions of dimension as low as 20 for VGAE and 30 for node2vec. On the contrary, we observe that the PVFs require dimension of at least 70 to reach comparable
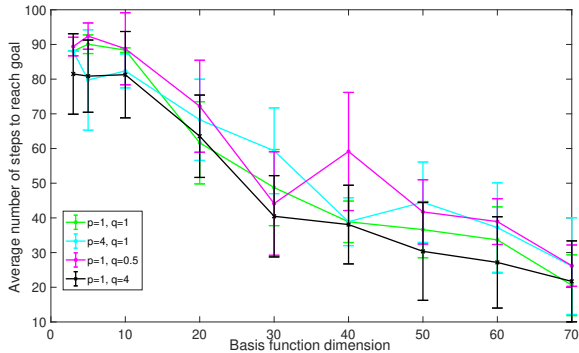
performances on the two-room domain and dimension of 50 on the obstacle-room domain.

We observe that the sampling strategy used in node2vec has a significant impact on the performance of the learned policy. Using grid search, we find that the optimal value of the parameters p and q that guide the random are 1 and 4 respectively. We show the performances of node2vec with selected values of p and q in Figure 6. When $p < q$ and $q > 1$, the strategy is biased to encourage walks to backtrack a step and to visit nodes that are close to the current node in the walk. Therefore, it leads to walks that approximate a breadth-first search behavior, gathering a local view of the underlying graph with respect to the starting node. On the other hand, when $p > q$ and $q < 1$, the walk approximate a depth-first search behavior and lead to more outward exploration. [Grover and Leskovec, 2016] show that the former type of sampling strategy allows to reflect structural equivalences of nodes whereas the second type allows to capture homophily within the network. Figure 6 suggests that for approximating value functions, structural equivalence plays a more important role than homophily.

4.3 Additional Results

In order to investigate whether we can expect a similar behaviour in larger environments, we consider a 100 by 50 three-room environment (similar to the two-room environment but with two interior walls, with the upper wall having the opening more on the right and the lower wall having the opening more on the left). We construct the graph from 500 collected samples of length at most 100 and derive the PVFs and the node2vec embeddings. For each of these basis function, we solve the linear approximation problem in
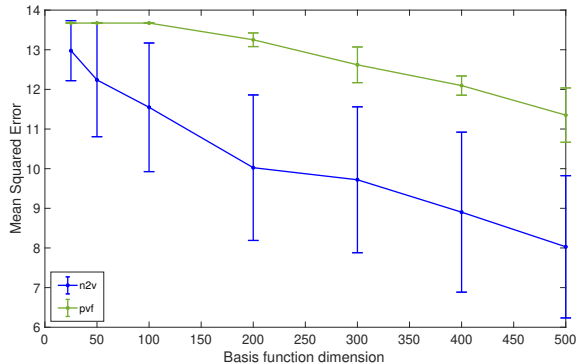
Figure 7: Mean squared error of value function approximation. On the x axis we make the dimension of the basis functions vary.

the least-square sense by minimizing the following loss function with respect to the parameter $\boldsymbol{\theta}$. The optimal value function is computed via value iteration [Montague, 1999]:

$$L(\theta) = \frac{1}{|S|} \sum_{s \in S} \left(V(s) - \sum_{i=1}^{d} \theta_i \phi_i(s) \right)^2$$

Figure 7 shows the gain of adopting node2vec feature learning in reinforcement learning in high dimensional state space.

4.4 Main Findings and Future Work

We summarize below the main findings of our work.

- The smoothness assumption of the value function on an estimated unweighted graph does not necessarily hold.

- Using basis functions that automatically learn to embed the geometry of the graph induced by the MPD can lead to improved performance over the proto-value functions.

- Such embedding models need to capture the structural equivalence of the nodes while preserving the local properties of the graph.

- Under sampling strategies that satisfy the requirements of the previous point, Node2vec [Grover and Leskovec, 2016] outperforms the commonly used proto-value functions.

- The Variational Graph Auto-Encoder, which is more complex than node2vec in terms of the number of trainable parameters, leads to minor performance improvement compared to node2vec.

These findings encourage the further study of representation learning on graphs for achieving efficient and accurate policy learning for reinforcement learning. In particular, the question of scalability in large or continuous state space arises. Future work includes analyzing to what extend one can efficiently learn good embeddings with limited samples in very large state spaces. Another interesting open question in this direction, is to investigate whether good representations can be inferred for states that have never been visited.

Naturally, future work should also aim at further improving the quality of the embeddings for solving reinforcement learning problems. A possibility would be to make use of the reward observed during the sample collection phase to build features that are not only based on state transitions, but capture reward information as well.

5 CONCLUSION

In this work, we have studied the representation policy iteration algorithm with a modified representation learning phase that allows to use any method for computing the basis functions in the linear value approximation. We investigated several models for learning high quality node embeddings that preserve the geometry of the graph induced by the Markov decision process. We compared the performance of several representation learning method in the context of value function approximation. Finally, we observe that embedding methods that are designed to capture the global structural geometry of the graph while preserving local properties do well at approximating the value function in low feature space dimensions, significantly outperforming the commonly considered proto-value functions for this task.

References

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, 1st edition.

[Donnat et al., 2018] Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. (2018). Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1320–1329, New York, NY, USA. ACM.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 855–864.

[Hamilton et al., 2017] Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[Kipf and Welling, 2016a] Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

[Kipf and Welling, 2016b] Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *CoRR*, abs/1611.07308.

[Konidaris et al., 2011] Konidaris, G., Osentoski, S., and Thomas, P. (2011). Value Function Approximation in Reinforcement Learning using the Fourier Basis. *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385.

[Lagoudakis, 2003] Lagoudakis, M. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

[Mahadevan, 2007] Mahadevan, S. (2007). Learning Representation and Control in Markov Decision Processes: New Frontiers. *Foundations and Trends® in Machine Learning*, 1(4):403–565.

[Mahadevan and Maggioni, 2006] Mahadevan, S. and Maggioni, M. (2006). Value function approximation with diffusion wavelets and laplacian eigenfunctions. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 843–850. MIT Press.

[Mahadevan and Maggioni, 2007] Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *J. Mach. Learn. Res.*, 8:2169–2231.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Montague, 1999] Montague, P. (1999). Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G. *Trends in Cognitive Sciences*, 3(9):360.

[Parr et al., 2008] Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 752–759, New York, NY, USA. ACM.

[Parr et al., 2007] Parr, R., Painter-Wakefield, C., Li, L., and Littman, M. (2007). Analyzing feature generation for value-function approximation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 737–744, New York, NY, USA. ACM.

[Petrik et al., 2010] Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010). Feature selection using regularization in approximate linear programs for markov decision processes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 871–878.

[Ribeiro et al., 2017] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 385–394, New York, NY, USA. ACM.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054.