Learning State-Action Basis Functions for Hierarchical MDPs

Sarah Osentoski SOSENTOS@CS.UMASS.EDU
Sridhar Mahadevan MAHADEVA@CS.UMASS.EDU
University of Massachusetts Amherst, 140 Governor's Drive, Amherst, MA 01002

Reinforcement Learning, Semi-Markov Decision Processes, Basis Function Learning, Manifold Learning

Abstract

This paper introduces a new approach to action-value function approximation by learning basis functions from a spectral decomposition of the state-action manifold. This paper extends previous work on using Laplacian bases for value function approximation by using the actions of the agent as part of the representation when creating basis functions. The approach results in a nonlinear learned representation particularly suited to approximating action-value functions, without incurring the wasteful duplication of state bases in previous work. We discuss two techniques to create state-action graphs: off-policy and on-policy. We show that these graphs have a greater expressive power and have better performance over state-based Laplacian basis functions in domains modeled as Semi-Markov Decision Processes (SMDPs). We present a simple graph partitioning method to scale the approach to large discrete MDPs.

1. Introduction

Recent work has focused on automatic basis construction for approximating value functions in Reinforcement Learning (Mahadevan, 2005; Keller et al., 2006; Smart, 2004). These techniques are designed to help solve large MDPs by constructing basis functions for value function approximation that compactly represent the topology of the state space. However, it is often desirable to approximate the action value function, $Q(s, a)$, rather than the state value function, $V(s)$. Approximating Q requires basis functions that are defined over state action pairs; previous approaches use techniques such as copying to transform their basis functions, defined over states, to basis functions that may be used to approximate Q. In this paper we propose incorporating actions into the embedding by creating representations in state-action space.

We specifically build upon the approach of using the graph Laplacian for constructing basis functions (Mahadevan, 2005). This is appealing since the basis functions are automatically constructed from the agent's experience in the domain. The approach consists of three steps: forming a graph where states are vertices and edges represent transitions to adjacent states, calculating the Laplacian of the graph, and computing the k smoothest eigenvectors of the Laplacian. In order to create basis functions over state-action pairs the eigenvectors of the state space graph Laplacian are copied for every action, zeroing out the bits corresponding to actions that were not performed.

The previous approach treats all actions as if they are equivalent, which is often not the case. Instead we create a graph in which the vertices are state action pairs, thus the resulting eigenvectors are already defined over state action pairs. The embeddings of this graph are in a different space: the similarity between state action pairs is dependent upon the actions that the agent takes in a state. States can now be differentiated; some actions in a state are more similar than others. Our technique is also capable of capturing smoothness in state-action space. This will not necessarily happen by copying basis function created in state space.

Our approach also benefits from the fact that it does not require copying because less basis functions are created. This is especially important in domains where the number of actions available in each state varies significantly; the eigenvector for a state must be copied for all possible actions, even those not available in the state. Embeddings created using these graphs are also able to differentiate between actions when several actions with different costs or durations lead from state s to state s'. In state graphs these differences cannot be modeled and

Appearing in *Proceedings of the 24^{th} International Conference on Machine Learning*, Corvallis, OR, 2007.

would be averaged or totally ignored.

Although state-action graphs provide significantly greater expressive power over state space graphs, they come at a price of potentially greater computational overhead. State-action graphs are significantly larger than their counterparts, and are also directed: this makes their construction and spectral analysis more intricate. We explore these tradeoffs in domains modeled as SMDPs. We select SMDPs for the following reasons: actions in these domains will vary greatly and the number of actions available in each state often varies.

2. SMDPs and SMDP Q-learning

We first briefly review Markov decision processes (MDPs). An MDP is defined as $M = (S, A, P^a_{ss'}, R^a_{ss'})$. S is the set of states, A is the set of actions, $P^a_{ss'}$ is the transition model, specifiying the probability of transitioning from state s to s' when action a is used, and $R^a_{ss'}$ is the reward model. Semi-Markov Decision Processes (SMDPs) differ from MDPs in that actions are no longer assumed to take a single time step and may have varied durations.

We use the option framework (Sutton et al., 1999) to model these temporally extended actions. An option o is defined as tuple $< I, \pi, \beta >$ where I is the initiation set which contains the states where the option may be initiated, π is the option policy which determines how the option will select actions or other options for execution, and β is the termination condition which gives the probability of option termination after each action in the world. The optimal option value function $Q^*(s, o)$ satisfies the Bellman equation

$$Q^*(s, o) = r^o_s + \sum_{s'} P^o_{ss'} \max_{o' \in O_{s'}} Q^*(s', o')$$

where $r^o_s = E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{K-1} r_{t+K} | \mathcal{E}(o, s, t)\}$. γ is the discount factor, $t + K$ is the time at which o terminates, and $\mathcal{E}(o, s, t)$ is the event of o being initiated in state s at time t.

We approximate Q as $\hat{Q}^\pi(s, o|w) = \sum_{j=1}^k \phi_j(s, o) w_j$ where Φ is a matrix with $|S| \times |A|$ rows and k columns, each column storing a basis function ϕ_j, and w_j are the weights.In order to learn $\hat{Q}$ we use SMDP $Q(\lambda)$-learning (Precup et al., 2000). This method uses an ϵ-greedy policy for action selection and accumulating eligibility traces. Traces are set to zero when a random action is taken. The update rule for the weights is given as

$$w_{t+K} = w_t + \alpha \delta_{t+K} \vec{e}_{t+K}$$

where

$$\delta_{t+K} = r_{t+K} + \gamma^{t+K} \max_{o \in O_{s_{t+K}}} \hat{Q}_t(s_{t+K}, o) - \hat{Q}_t(s_t, o_t),$$

$$\vec{e} = \gamma^{t+K} \lambda \vec{e}_{t+K} + \nabla_{\vec{w}_t} \hat{Q}_t(s_t, o_t), \text{ and } \vec{e}_0 = 0.$$

3. Directed Graph Laplacian

State-action graphs are inherently directed, and require a different graph Laplacian operator from undirected graphs. In this section we give a brief summary of spectral decomposition of the Laplacian on directed graphs; a more in depth analysis can be found in (Chung, 2005; Johns & Mahadevan, 2007). We first review the spectral decomposition of the Laplacian on undirected graphs (Chung, 1997). A weighted undirected graph is defined as $G_u = (V, E_u, W)$ where V is set of vertices, E_u is the of set edges, and W is the set of weights w_{ij} for each edge $(i, j) \in E_u$. If an edge does not exist between two vertices it is given a weight of 0. The valency matrix, D, is a diagonal matrix whose values are the row sums of W. The combinatorial Laplacian is defined as $L_u = D - W$ and the normalized Laplacian is defined as $\mathscr{L}_u = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.

A weighted directed graph is defined as $G_d = (V, E_d, W)$. The major distinction between the directed and undirected graph is the non-reversibility of edges. A directed graph may have weights $w_{ij} = 0$ and $w_{ji} \neq 0$. This is not possible in an undirected graph. This difference is the key reason that directed graphs are preferable for encoding policy graphs. In order to define the graph Laplacians on G_d we must first introduce the Perron vector, ψ. A random walk on G_d defines a probability transition matrix $P = D^{-1}W$. The Perron-Frobenius Theorem states that if G_d is strongly connected then P has a unique left eigenvector ψ with all positive entries such that $\psi P = \rho \psi$ where ρ is the spectral radius. ρ can be set to 1 by normalizing ψ such that $\sum_i \psi_i = 1$. A more intuitive way of thinking of ψ is as the long-term steady state probability of being in any vertex over a long random walk on the graph.

There is no closed-form solution for ψ; however, there are several algorithms to calculate it. The power method (Golub & Loan, 1989), is an approach to iteratively calculate ψ that starts with an initial guess for ψ, uses the definition $\psi P = \psi$ to determine a new estimate and iterates. Another technique is the Grassman-Taksar-Heyman (GTH) algorithm. This technique uses a Gaussian elimination procedure designed to be numerically stable. The naive GTH implementation runs in $O(n^3)$, but this can be improved in $O(nm^2)$ if P is sparse. Other techniques, such as Perron complementation (Meyer, 1989), have been

introduced to speed up convergence.

The graph Laplacians for the directed graph are defined as (Chung, 2005)

$$L_d = \Psi - \frac{\Psi P + P^T \Psi}{2}$$
$$\mathscr{L}_d = I - \frac{\Psi^{1/2} P \Psi^{-1/2} + \Psi^{-1/2} P^T \Psi^{1/2}}{2}$$

where Ψ is a diagonal matrix with entry $\Psi_{ii} = \psi_i$. To find basis functions on directed state-action graphs, we compute the k smoothest eigenvectors of L_d or $\mathscr{L}_d$. These eigenvectors form Φ and can be used in a learning algorithm as described in Section 2. A description of the effects of using L_d and $\mathscr{L}_d$ as state basis functions for solving MDPs can be found in (Johns & Mahadevan, 2007).

The directed Laplacian requires a strongly connected graph, however graphs created from an agent's experience may not have this property. In order to ensure that this property exists we use a teleporting random walk (Page et al., 1998). With probability η the agent acts according to the transition matrix P and with probability $1-\eta$ teleports to any other vertex in the graph uniformly at random. This assumption is not built into the domain and is only used for the purpose of creating ψ and performing the spectral decomposition.

4. Manifold Construction

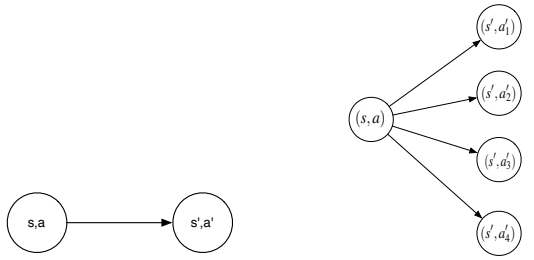
Table 1 presents different ways of creating the weight matrix W for state and state-action graphs. We explain below why these weightings were chosen. $W(i,j)$ is the weight for the transition between state i and state j. $W(x,y)$ is the weight in the state-option graph where x is the vertex corresponding to state i and one of its available options and y is the vertex for state j and one of its available options. $\overline{time(i,j)}$ is the average duration of the option transitioning from state i to state j. $count(i)$ is the number of times state i is observed in the samples and $count(i,j)$ is the number of times the transition from state i to state j is observed. The weight matrix W is also designed to take the duration of the temporally extended actions of the SMDP into account by weighting each option edge by the inverse of the option's average duration. We use the inverse because the Laplacian treats W as a similarity matrix rather than a distance matrix.

State-option graph	$W(y,z) = \frac{1}{\overline{time(i,j)}} \frac{count(i,j)}{count(i)}$
State graph	$W(i,j) = \frac{1}{\overline{time(i,j)}}$

Table 1. Weightings Used for SMDP Graphs.

4.1. Off-Policy vs. On-Policy Graph Construction

Two techniques, shown in Figure 1, may be used to create state-action graphs. The first, on-policy graph creation, connects (s,a) to (s',a') if the agent is in state s, takes action a, transitions to state s' and then selects action a'. The second, off-policy graph creation, connects (s,a) to (s',a') if the agent is in state s, takes action a, transitions to s' and $a' \in A_{s'}$ where $A_{s'}$ is the set of actions available in state s'. Self loops are not included in either type of graph creation. On-policy graph creation can be used to model the current policy that the agent is performing while the off-policy graph will encode the underlying MDP. When the agent is executing a random walk the two techniques will converge to the same graph however, the off-policy method requires less exploration to construct the graph. State-action graphs can be significantly impacted



(a) On-policy graph creation only adds a single edge based on the exact experience of the agent.

(b) Off-policy graph creation adds edges for all actions available in state s'.

Figure 1. Two techniques for creating state-action graphs.

when the environment is stochastic. If this is not corrected unlikely transitions will have similar weights as likely transitions. In order to compensate for this, as shown in Table 1, we keep track of the frequency of transitions during the exploration period and then multiply the edge weight by this number. While state-action graphs can more accurately model the environment they add complexity due to the fact that they have directed edges and have an increased number of vertices. Constructing state-action graphs based on a random walk can be inefficient. We used off-policy graph creation to reduce the number of samples needed to create the graph. We also modified the random walk so that the agent selected the action that has been least frequently used in a given state.

5. Domain and Experiments

5.1. Four Room Grid World

In order to illustrate our technique we used a four room gridworld shown in Figure 2 (Sutton et al., 1999) This domain consists of 169 states of which 104 are free states

(states that are not a wall). In any free state the agent can perform one of four primitive actions: north, south, east or west. There is a 10% probability that an action will fail and the agent will remain in the same location. If the agent moves into a wall it remains in the same location. Rewards are zero on all state transitions except transitions into the goal state when the agent receives a reward of 100. Two

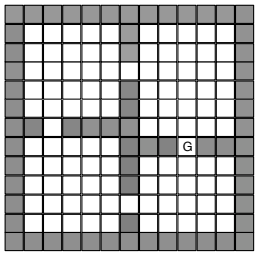
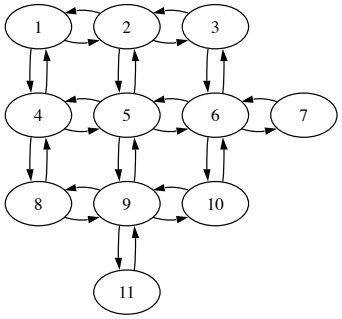


Figure 2. Four room gridworld.

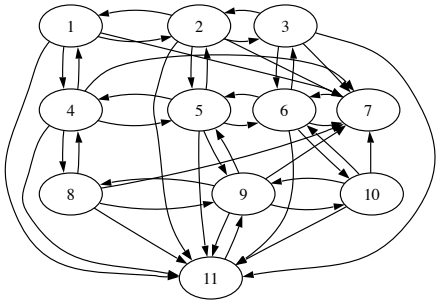
hallway options are provided in each of four rooms. These options allow the agent to navigate from any location within one room to one of the two hallway states that lead out of that room. The initiation set, I, is comprised of all the states within the room. A hallway option's policy is optimal and cannot be terminated once selected until it reaches the goal state. Hallway states do not have hallway options available to them and are not in the initiation set of any of the hallway options.

We consider a learning problem in this domain. The agent must use the 8 multi-step hallway options and primitive actions to learn to reach the goal. We first allow the agent to explore the environment selecting from primitive actions and available options randomly. We used 2000 episodes with 50 steps per episode. We perform this exploration only once. The agent then builds the graph from these samples and computes the basis functions. We use SMDP Q(λ)-learning($\gamma = .9, \epsilon = .1, \alpha = .01$) as described in Section 2 to learn a policy that maximizes the agent's long-term reward. The agent starts in a random state at the beginning of each learning episode and the agent was allowed to select actions as long as the number of steps was less than 50 in a learning iteration (the agent may take slightly more than 50 steps if at timestep 49 it selected an option). The Q-function was initially set such that all values were zero. We then performed an initial learning step on the random samples to initialize the policy before performing SMDP Q(λ)-learning. The initial learning iteration did not prove to be very useful since the agent often spends much of its time wandering in one portion of the state space. Random actions only allow for a one-step backup since we are using Watkin's $Q(\lambda)$ and traces are cut off every time an exploratory action is taken.

In order to illustrate the structure of the graphs created in this domain we use a smaller domain in which the upper left room has only 9 states. The domain is exactly identical to the four room gridworld except that it is smaller. We show the graphs for only the upper left room. Figure 3(a) shows the state graph where the agent can only execute primitive actions. Figure 3(b) illustrates the case when the agent has access to options. The options introduce long edges going from each state to states 7 and 11. Figure 4 shows the state-action graph when the agent



(a) State graph showing transitions when the agent has only primitive actions.

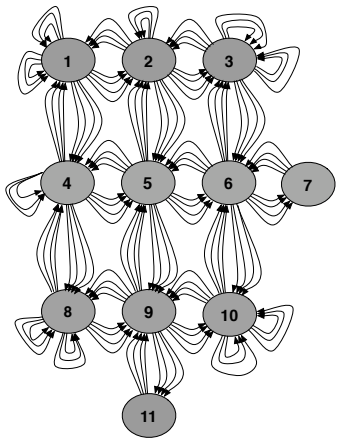


(b) State graph showing transitions when the agent also has access to hallway options.

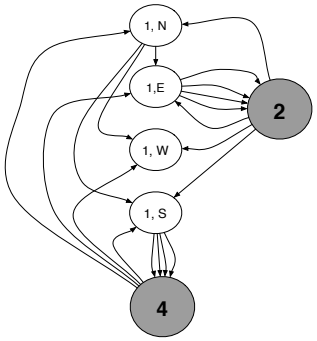
Figure 3. The state graphs created for the upper left hand room in a smaller version of the four room gridworld.

can only execute primitive actions. Figure 4(a) shows the global topology of the graph. This graph is similar to the topology of the state graph. However, each node in this figure represents the 4 state-action pairs for each state. Figure 4(b) shows the state-action pairs specifically for state 1. Figure 5 shows the state-action pairs for state 1 when the agent has access to the hallway options. Two nodes are added to represent the new state-action pairs. Transitions from these nodes will lead to the 4 state-action nodes of the hallway states associated with the transition.

Figure 6 shows the vertices of the graphs of the full domain using the second and third eigenvectors as co-



(a) State-action graph showing transitions when the agent has only primitive actions.



(b) Close up of transitions associated with nodes for state-action pairs for state 1.

Figure 4. The state-action graphs created for a small room.

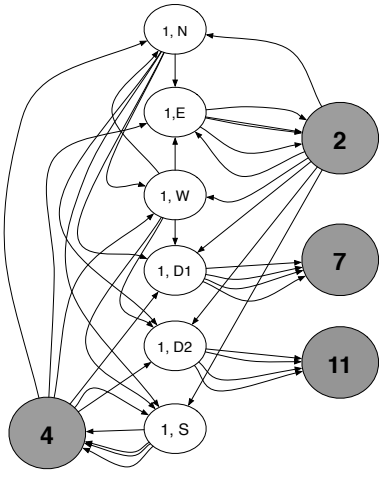
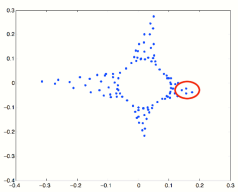
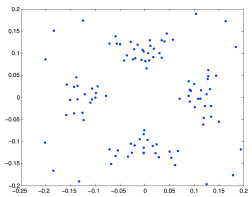


Figure 5. Transitions associated with nodes for the state-action pairs for state 1 when the doorway options are available.

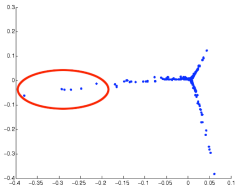
ordinates. The agent had access to the hallway options for these graphs. These figures show that the technique is grouping the graphs into four clusters corresponding to the four rooms. The tip of each of these groups is the corners of the domain. While Figure 6(c) appears to be the one outlier, the center of the graph has a shape similar to Figure 6(a). Figure 7 provides a zoomed in view of the vertices of the graph for the states in the upper right-hand corner of the domain as seen in Figure 7(a). Figure 7(b) shows the embedding of the states while Figure 7(c) shows the embedding of the state-action pairs. The areas zoomed in on are indicated by circles in Figure 6. As can be seen in Figure 7(c) action 1 (North) and 2 (East) are located on the same point. This is a desirable result as state 25 is located at the top right corner of the grid and both actions will transition back to state 25. State-action pairs that have similar transitions are also placed near to each other, (25,3(South)), (38,1) and (25,4(West)), (24,2), are examples of this. This proximity in the embedding space is highly desirable as it yields good generalization across state-action space.



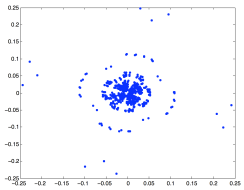
(a) The embedding of the combinatorial Laplacian of the state graph



(b) The embedding of the normalized Laplacian of the state graph.



(c) The embedding of the combinatorial Laplacian of the state-action graph.

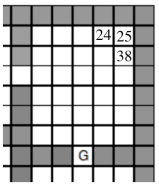


(d) The embedding of the normalized Laplacian of the state graph.

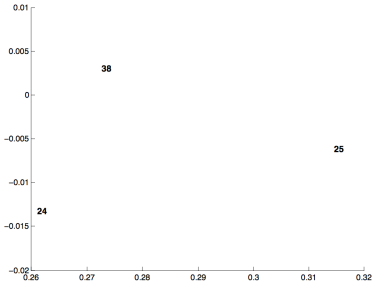
Figure 6. The embeddings of the graph Laplacians using the second and third eigenvectors for the four room gridworld domain when options are available to the agent.

5.2. Eight Room Gridworld

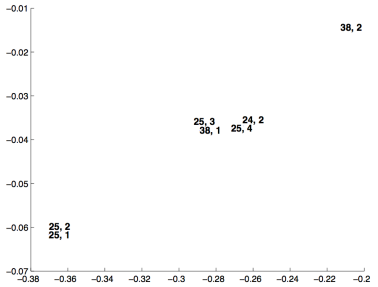
We also ran experiments on an eight room gridworld shown in Figure 8. This domain consists of 325 states of which 210 are free states. In any free state the agent can perform one of four primitive actions: north, south, east or west. There is a 10% probability that an action will fail and the agent will remain in the same location. If the



(a) The right hand corner of the four room gridworld with the corner states labeled.



(b) The embedding of the combinatorial Laplacian on the state graph.



(c) The embedding of the combinatorial Laplacian on the state-action graph.

Figure 7. A closer look at the embeddings of the combinatorial graph Laplacian

agent moves into a wall it remains in the same location. Rewards are zero on all state transitions except transitions into the goal state when the agent receives a reward of 100. Hallway options are provided in each of four rooms. These

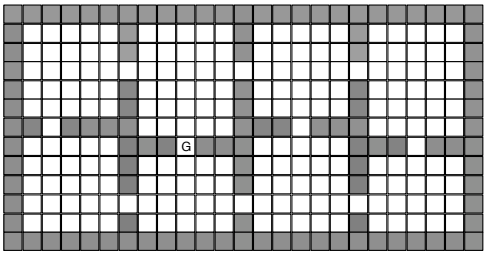


Figure 8. Eight room gridworld.

options allow the agent to navigate from any location within one room to one of the hallway states that lead out of that room. Rooms adjacent to 3 doorways have 3 hallway options and rooms adjacent to 2 doorways have 2 hallway options. The initiation set, I, is comprised of all the states within the room. A hallway option's policy is optimal and cannot be terminated once selected until it reaches the goal state. Hallway states do not have hallway options available to them.

The learning problem in this domain is that the agent must use the 20 multi-step hallway options and primitive actions to learn to reach the goal. We first allow the agent to explore the environment selecting from primitive actions and available options randomly. We used 4000 episodes with 50 steps per episode. We perform this exploration only once. The agent then builds the graph from these samples and computes the basis functions. We use SMDP Q(λ)-learning($\gamma = .9, \epsilon = .1, \alpha = .01$). The agent is allowed 100 steps per learning episode and the Q-function is initialized to zero.

6. Results

We performed experiments to compare the two graph Laplacians on both state and state-action graphs. In these experiments we systematically varied the number of basis functions used in function approximation. In experiments using state graphs we varied the number of basis functions from 24 to 120 in steps of 12 and from 144 to 1200 in steps of 24. In experiments using state-action graphs we varied the number of basis functions from 3 to 10 in one step increments and from 20 to 600 in increments of 10. The results of each experiment was averaged over 200 trials and each experiment was performed for 300 learning iterations.

Figure 9 compares the number of steps taken by the agent to reach the goal when using the two types of graph Laplacians on both state and state-action graphs. The performance of the normalized and combinatorial graph Laplacians was similar on both the state and state-option graphs, thus we plot only results using the normalized Laplacian. The best performance was in experiments using the state-option graphs with 260 basis functions (out of 616 possible basis functions). A similar number of basis functions created from the graph Laplacians of state graphs does not yield similar performance. Instead performance using about 264 basis functions created from the state graph was similar to performing table lookup. Using more basis functions did not improve performance noticeably. All techniques using options outperformed experiments where the agent had access only to primitive actions.

Basis functions created from state-action graphs performed the best; we were not able to achieve similar performance with basis functions derived from the state graph. We also performed experiments varying α however the results were not significantly changed. Basis functions created from state-action graphs continually outperform basis functions created from state graphs.

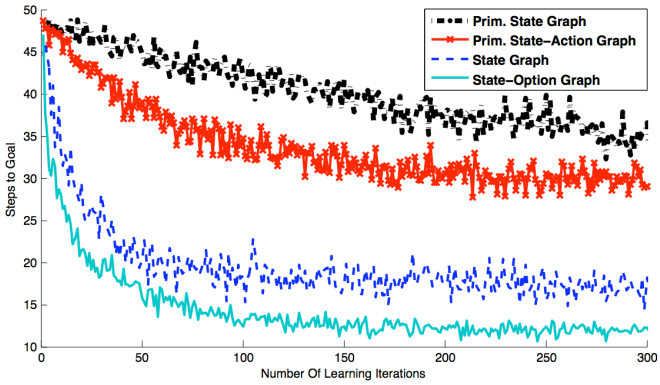


Figure 9. Steps to goal in the four room gridworld.

We performed similar experiments to compare the two graph Laplacians on both state and state-action graphs in the eight room gridworld. In these experiments we systematically varied the number of basis functions used in function approximation. The results of each experiment was averaged over 200 trials and each experiment was performed for 600 learning iterations. Figure 10 shows that once again the state-option graphs out perform the state graphs. In the eight room grid world we use 780 (out of 2520) basis functions on the state-option graph. The best results using a state graph required 2400 basis functions. Both results are shown using the normalized graph Laplacian.

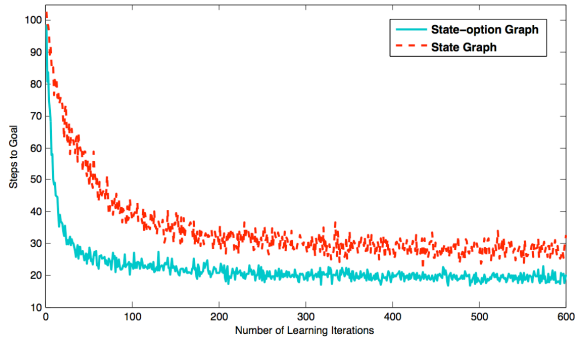


Figure 10. Steps to goal in the eight room gridworld.

6.1. Scaling

One concern with this approach is that state-action graphs grow significantly as the state-action space of the domain increases. We illustrate one approach to scaling using graph segmentation to break the graph into g smaller subgraphs. We use METIS (Karypis & Kumar, 1998), an automatic graph partitioning algorithm, to partition the graph. METIS requires undirected graphs as its input so we symmetrize the graph for partitioning but build the basis functions on the partitioned directed graphs. We perform spectral decomposition to get k' eigenvectors from each of g smaller subgraphs where $k' = k/g$. While the number of basis functions remains unchanged, the basis functions become extremely sparse allowing the matrix to be easily compressed. Each experiment was performed for 600 learning iterations and results are averaged over 30 trials. We show results with basis functions created from the normalized graph Laplacian on state-action graphs.

Figure 11 shows the result of scaling in the four room gridworld. This figure compares the result of using the full state-action graph and the partitioned state-action graph where $g = 4$. In this experiment we use 260 eigenvectors. 75% of the values in the basis function matrix created on the partitioned graph were zero.

When we attempted a similar approach on the eight room gridworld we were not able to achieve satisfactory performance. The best partitioning we found was with $g = 10$. However the agent converged to a solution that required on average 20 more steps to find the goal. Better results may be obtained with a more sophisticated algorithm. The technique presented here is a simplified version of the approach described in (Johns et al., 2007). In this work, the graph partitioning step is used to construct a permutation matrix that rearranges the rows and columns of the basis matrix. A separable least-squares method is then employed to construct a Kronecker factorization of the permuted basis matrix. Finally, the authors use a Markov chain Monte-Carlo algorithm to construct reversible stochastic sub-matrices from the Kronecker factors.

Other approaches to segmenting the graphs could also be used. States could be clustered based on actions available in the state. MDP homomorphisms could also be used partition the graphs. Previous work has used homomorphisms in the context of learning control in SMDPs (Ravindran & Barto, 2003). However, homomorphisms have not, to our knowledge, been used in basis function construction. Another improvement to this approach would be to perform the graph partitioning on the directed graph. This should lead to better partitions and thus better performance.

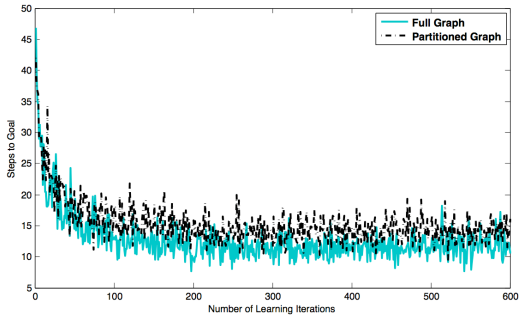


Figure 11. Comparison of basis functions constructed on a partitioned state-action graph vs the full graph for the four room gridworld.

7. Conclusions and Future Work

This paper explored the effectiveness of constructing basis functions for approximating action value functions from the spectral decomposition of the graph Laplacian on state-action graphs. The results show that basis functions defined over state-action graphs are able to improve performance over those defined over state graphs. We also presented a technique for scaling the approach using graph partitioning.

There are several avenues for future work. An experiment examining different weighting techniques for state option graphs needs to be carried out. We selected a simple technique; however, other weightings are possible such as the inverse of the discounted sum of the transition times. Another issue is how to extend this work to continuous domains. This extension has been carried through for spectral bases defined on state graphs (Mahadevan et al., 2006; Johns et al., 2007); however some modifications of this technique would be required to accommodate state-action graphs. Specifically, a distance metric between actions or state-action pairs must be specified. Another area for future work is building the graphs and basis functions incrementally. Currently the agent must explore the state-action space in order to create basis functions. An incremental method would allow the agent to begin learn control and representation simultaneously.

Acknowledgments

This research was supported in part by the National Science Foundation under grant NSF IIS-0534999. The authors thank the members of their Autonomous Learning Lab for their comments and assistance.

References

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

Chung, F. (2005). Laplacians and the Cheeger Inequailty for Directed Graphs. *Annals of Combinatorics*, *9*, 1–19.

Golub, G., & Loan, C. V. (1989). *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press. 2nd edition.

Johns, J., & Mahadevan, S. (2007). Constructing basis functions from directed graphs for value function approximation. *Proceedings of Twenty-fourth International Conference on Machine Learning (ICML)*.

Johns, J., Mahadevan, S., & Wang, C. (2007). Compact spectral bases for value function approximation using kronecker factorization. *National Conference on Artificial Intelligence (AAAI)*.

Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, *20*, 359–392.

Keller, P., Mannor, S., & Precup, D. (2006). Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning. *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY: ACM Press.

Mahadevan, S. (2005). Proto-Value Functions: Developmental Reinforcement Learning. *Proceedings of the 22nd International Conference on Machine Learning* (pp. 553–560). New York, NY: ACM Press.

Mahadevan, S., Maggioni, M., Ferguson, K., & Osentoski, S. (2006). Learning Representation and Control in Continuous Markov Decision Processes. *Proceedings of the 21st National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Meyer, C. (1989). Uncoupling the Perron Eigenvector Problem. *Linear Algebra and its Applications*, *114/115*, 69–94.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). *The PageRank Citation Ranking: Bringing Order to the Web* (Technical Report). Stanford University.

Precup, D., Sutton, R., & Singh, S. (2000). Eligibility traces for off-policy policy evaluation. *Proceedings of the 17th International Conference on Machine Learning* (pp. 759–766). Morgan Kaufmann.

Ravindran, B., & Barto, A. (2003). Smdp Homomorphisms: An Algebraic Approach to Abstraction in Semi Markov Decision Processes. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 03)* (pp. 1011–1016). AAAI Press.

Smart, W. (2004). Explicit manifold representations for value-function approximation in reinforcement learning. *Prceedings of the 8th International Symposium on Artificial Intelligence and mathematics*.

Sutton, R., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.