Proto-Transfer Learning in Markov Decision Processes Using Spectral Methods

Kimberly Ferguson

University of Massachusetts Amherst
Department of Computer Science

May 30, 2008

1 Introduction

The aim of transfer learning is to reuse behavior by using the knowledge learned about one domain or task to accelerate learning in a related domain or task. The new framework of proto-transfer learning transfers the representation rather than the behavior. This transfer of representation entails the reuse of eigenvectors learned from one graph in another. We explore how to transfer knowledge learned on the source graph to a similar graph by modifying the eigenvectors of the Laplacian of the source to be reused for the target. In this paper we explore solutions to transfer learning within reinforcement learning domains (Sutton & Barto, 1998) through spectral methods. This abstraction of representation is something that humans and animals do effortlessly, and is thus of great research importance to the machine learning community.

For example, a neuroscience study on primates has shown that a rhesus monkey (Ivan) is capable of using tools for multiple goals. Ivan learns to use a rake to retrieve a treat in different locations and to perform the same task with different tools (task transfer). Additionally, it has been shown that he can maneuver the rake over various barriers of his cage and even go around a corner to retrieve the treat (domain transfer) (Veino & Novak, 2003).

In past work, (Foster & Dayan, 2000) studied the task transfer problem by applying unsupervised, mixture model, learning methods to a collection of optimal value functions of different tasks in order to decompose and extract the underlying structure. Proto-value functions (PVFs) are a natural abstraction since they condense a domain by automatically learning an embedding of the state space based on its topology (Mahadevan, 2005). PVFs lead to the ability to directly transfer knowledge about domains and tasks, since they are constructed without taking reward into account.

We define *task transfer* as the problem of transferring knowledge when the

state space remains the same and only the reward differs. For task transfer, task-independent basis functions, such as PVFs, can be reused from one task to the next without modification. *Domain transfer* refers to the more challenging problem of the state space changing. This change in state space can be a change in topology (.i.e. obstacles moving to different locations) or a change in scale (.i.e. a smaller or larger domain of the same shape). For domain transfer, the basis functions will need to be modified to reflect the changes in the state space.

In this paper, we investigate task transfer in discrete domains by reusing PVFs in Markov decision processes (MDPs) with different rewards functions and different domain dynamics. For domain transfer, we introduce a systematic approach to modify the eigenvectors for abstraction in the new domain. We will show results applying the Nystrõm extension (Mahadevan et al., 2006) for interpolation of PVFs between MDPs of different sizes as well as Procrustes alignment (Gower, 1975). Previous work has been able to speed up learning when transferring behaviors between tasks and domain (Taylor et al., 2005), but we are able to transfer representation and reuse knowledge to learn just as well on a new task or domain. We extend upon our previous results (Ferguson & Mahadevan, 2006) to include task transfer where the world's dynamics change, and to improve the robustness of our domain transfer results by using a new technique.

2 Framework

2.1 Reinforcement Learning

Reinforcement learning systems contain four main elements: 1) a policy, 2) a reward function, 3) a value function, 4) a model of the environment. The policy is a mapping of what action an agent should take in each state of the environment. We are ultimately trying to optimize the policy so that the agent can act optimally in accordance to the reward function. The reward function defines the goal of the system, by assigning a numerical reward to each state. The value function is the discounted estimate of reward for each state based on likely future actions. So, the value of a state is the amount of reward an agent can expect to accumulate when starting from that state. The model of the environment of a reinforcement learning task determines the agent's next state given a current state and action. The dynamics of the model include details such as if the environment is deterministic or with what probability (which we refer to as %-stochastic) an agent attempting to take a particular action will take a random action instead. A reinforcement learning problem that satisfies the Markov property is called a Markov decision process.

2.2 Markov Decision Process

A Markov decision process (MDP) $M = \langle S, A, P^a_{ss'}, R^a_{ss'} \rangle$ is defined by a set of states $S \subset \mathbb{R}^d$, a set of discrete actions A, a transition model $P^a_{ss'}$ specifying

the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R^a_{ss'}$ specifying a scalar cost or reward. The state-action value function $Q^\pi(s,a)$ of any policy π can be found for all state-action pairs by solving the linear system of the Bellman equations:

$$Q^\pi(s,a) = R^a_{ss'} + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a', s') Q^\pi(s', a'). \quad (1)$$

2.3 Obstacle Gridworld

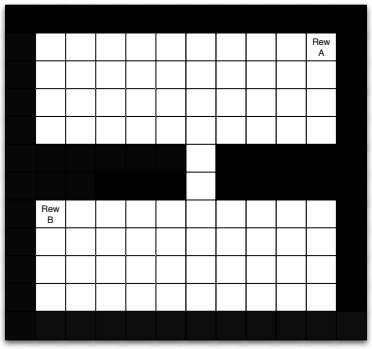


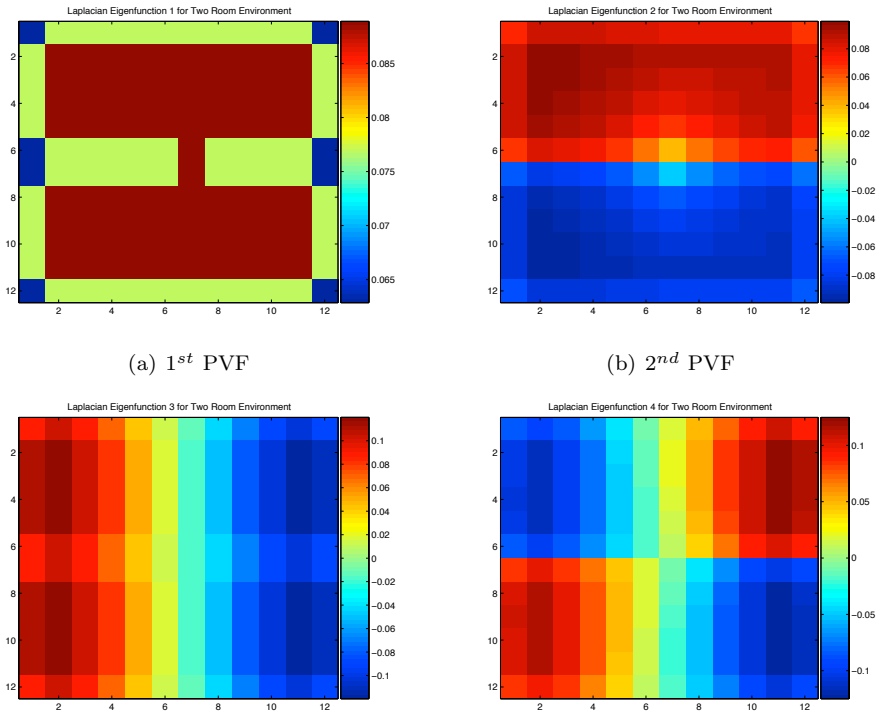
Figure 1: 12x12 two-room gridworld with various reward functions.

The reinforcement learning task that we will be experimenting on is the obstacle gridworld–in particular, a two-room gridworld (see Figure 1). Sometime we will use the gridworld with no obstacles, also known as a one-room gridworld. The possible actions are North, South, East, and West. If an agent tries to move into a wall, it will remain in the same state. To estimate the value function, we collect samples using a random walk of a maximum of 200 episodes, each with a maximum of 150 steps and random start state. Non-goal states have zero reward; the goal has reward of 100. This is a discrete domain with 144 states. However, we are also interested in much larger discrete domains, as well as more complex continuous domains.

2.4 Function Approximation

In large discrete or continuous domains where not all states will be visited enough to get a good estimate of a value function (some states may not be visited at all). Function approximation attempts to generalize from the estimated value functions of some states, an approximation over the entire state space. There are many different approaches for function approximation. Proto-value functions approximate the value function using spectral methods.

2.5 Proto-value Functions



(a) 1^{st} PVF (b) 2^{nd} PVF

(c) 3^{rd} PVF (d) 4^{th} PVF

Figure 2: Example PVFs of the 12x12 two-room gridworld (Figure 3(c)). Notice how the PVFs capture the structure inherent to the state space.

Proto-value functions (PVFs) are an orthonormal basis spanning all value functions on a state space. PVFs are constructed as follows: 1) from an initial random walk, create an adjacency matrix which reflects the topology of the state space; 2) compute the graph Laplacian of the adjacency matrix; 3) use the smoothest k eigenvectors (ranked by eigenvalue) of this graph Laplacian as PVFs. Thus, PVFs are a bases which respect the topology of the state space (See Figure 2).

More formally, let $G = (V, E, W)$ denote a weighted undirected graph with vertices V, edge set E and weights w_{ij} on edge $(i, j) \in E$. The degree of a vertex v is denoted as d_v. The adjacency matrix A can be viewed as a binary weight matrix describing the connectivity of the graph. Let D be the valency matrix, in other words, a diagonal matrix whose entries are the row sums of A. The nomalized Laplacian $\mathcal{L}$ of the graph G is defined as $\mathcal{L} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$, where the states are the vertices, and edges connect states that are adjacent in

the state space (i.e. a state that can be reached from that state). Specifically,

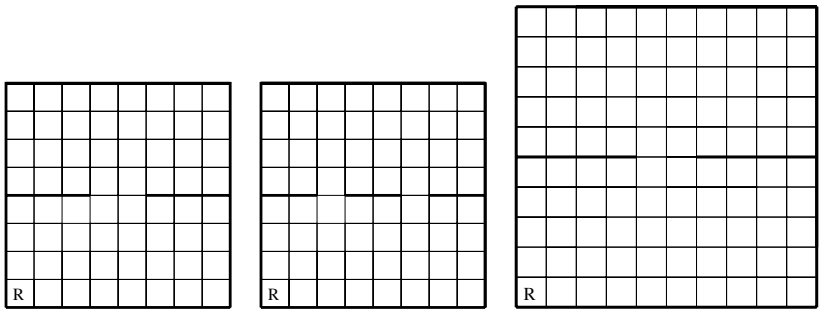
$$\mathcal{L}(u,v) = \begin{cases} 1 - \frac{1}{d_v} & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$\mathcal{L}$ is a symmetric self-adjoint operator, and its spectrum (eigenvalues) lie in the interval $\lambda \in [0, 2]$. PVFs are the eigenvectors $\phi_i(\mathcal{L})$, such that $\mathcal{L}\phi_\rangle = \lambda_\rangle \phi_\rangle$.

2.6 Task Transfer

For the task transfer problem, the graph Laplacian $\mathcal{L}$ of source graph G_S and target graph G_T are the same, since only the reward function or model dynamics have changed, and their adjacency matrix A is the same for both graphs. Thus the eigenvectors $\phi_i(\mathcal{L})$ of G_S can be transferred to G_T. An example of task transfer of reward is shown in Figure 1; where reward A is the source domain and reward B is the target. We will also examine the case of task transfer of dynamics.

2.7 Domain Transfer (topology)



(a) 8x8 domain transfer source. (b) 8x8 topological domain transfer target. (c) 10x10 scaling domain transfer target.

Figure 3: Two-room gridworld examples of topological and scaling domain transfer.

For the domain transfer problem where the shape of the state space changes, the connectivity of the graph G_S is different from that of G_T and the adjacency matrix of the target A_T is the adjacency matrix of the source A_S, perturbed by some matrix E, i.e. $A_T = A_S + E$. Thus, we can view the differences in the corresponding Laplacians of the source and target, $\mathcal{L}_S$ and $\mathcal{L}_T$ as:

$$\mathcal{L}_S = D_S^{-\frac{1}{2}}(D_S - A_S)D_S^{-\frac{1}{2}}$$
$$\mathcal{L}_T = D_T^{-\frac{1}{2}}(D_S - [A_S + E])D_T^{-\frac{1}{2}}$$

We are currently exploring matrix perturbation theory to quantify how the eigenvalues and eigenvectors $\phi_i(\mathcal{L}_T)$ change based on the perturbation E (i.e. changes in the connectivity of the graph). An example of topological domain transfer is shown in Figure 3; Figure 3(a) is the source domain and Figure 3(b) is the target.

2.8 Domain Transfer (scale)

The domain transfer problem where the size of the state space changes, focuses on the expansion of the connectivity of the graph G_S, where the pattern of the adjacency graph A_S is retained in A_T, while the sizes of the matrices differ. We use various techniques to extend the eigenfunctions $\phi(\mathcal{L}_S)$ computed on A_S to the new states of A_T to create $\phi(\mathcal{L}_T)$. An example of scaling domain transfer is shown in Figure 3; Figure 3(a) is the source domain and Figure 3(c) is the target.

3 Algorithmic Details

We use *source* and *target* to describe the domain we transfer knowledge from and to, respectively. We include the term *pure* when the PVFs are created from and used for learning on the same (source) graph, while *transfer* will refer to the case in which the PVFs are created on a (source) graph and transferred to be used for learning on another (target) graph. Least-squares Policy Iteration (LSPI) is used to learn the control policy, where the underlying subspace for approximating the value function is spanned by the learned PVFs.

4 Experimental Results

4.1 Task Transfer (Reward)

These *task transfer* experiments investigate transfer learning using PVFs, where the state space and basis functions are constant, but the *reward function is varied* . Since this method creates basis functions based on the actual topology of the state space, it is a natural solution to this task transfer problem.

In Table 1 we see that transferring the basis functions from one grid (Exp 1.a, with reward in the upper right-hand corner) to grids with different reward functions (Exp 1.b, with reward in the lower left-hand corner and Exp 1.c, with reward in the middle) does not affect performance. Thus we have shown that proto-value functions are reward independent. These 12x12 gridworlds all have 144 states (with the same topology), we use the 'lsqfast' algorithm, a discount of 0.9, 130 eigenvectors, and allow 20 iterations. We collect samples using a random walk of a maximum of 200 episodes, each with a maximum of 150 steps and random start state. During testing, the learned policy is evaluated allowing a maximum of 50 steps, and averaged over 20 runs. The reward function assigns zero reward to all states except for the goal which has reward of 100.

`Proto-transfer` $(dom_S, dom_T, S_{\{S,T\}}, J, N, \epsilon, k, P)$:

1. **Representation Learning Phase**: Perform a random walk of J trials, each of maximum N steps on the source domain dom_S, and store the states visited in the dataset $\mathcal{D}_S$.

   (a) *Create PVFs for the source domain*: Build an undirected weighted graph G from $\mathcal{D}$ where edges can be inserted between a pair of points x_i and x_j if , x_j is among the k nearest neighbors of x_i and all edges have weight 1. Construct the normalized Laplacian $\mathcal{L}$ on graph G as in Equation 2.

   (b) Compute the k smoothest eigenvectors of $\mathcal{L}$ on the graph G, and collect them as columns of the basis function matrix Φ, a $S_S \times k$ matrix, where S_S is the number of states in the source. The embedding of a state action pair $\phi(s,a)$ where $s \in \mathcal{D}$ is given as $e_a \otimes \phi(s)$, where e_a is the unit vector corresponding to action a, $\phi(s)$ is the s^{th} row of Φ, and $\otimes$ is the tensor product.

2. **Control Learning Phase**: Perform a random walk of J trials, each of maximum N steps on the target domain dom_T, and store the states visited in the dataset $\mathcal{D}_t$. Initialize $w^0 \in \mathcal{R}^k$ to a random vector.
   Repeat the following steps:

   (a) *Transfer PVFs from source to target domain*: Set $i \leftarrow i+1$. For each transition $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}_T$, compute low rank approximations of matrix A and b as follows:

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

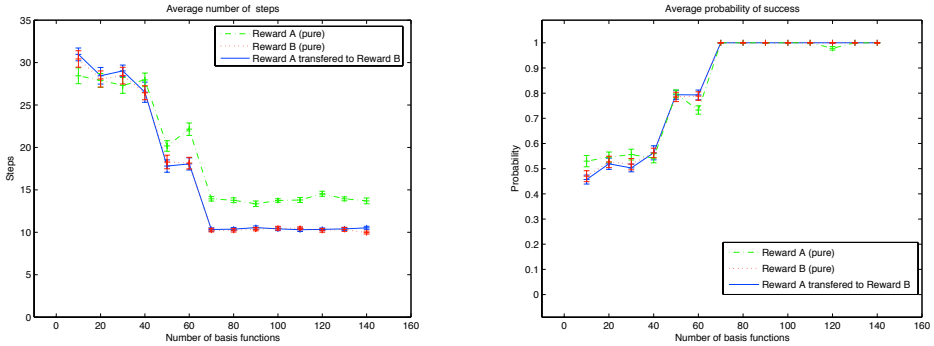
   where $\phi(s_t,a_t)$ is approximated using the Nyström extension (Equation **??**) when $s_t \notin \mathcal{D}_S$ (necessary for domain transfer only).

   (b) Solve the system $\tilde{A}w^i = \tilde{b}$

3. **until** $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
4. Return $\hat{Q}^\pi = \sum_i w^i \Phi$ as the approximation to the optimal value function.

Figure 4: Pseudo-code of the proto-transfer algorithm for both task and domain transfer learning.

The results in Figures 5 are for transfer experiments set up the same as above except for the following: We use a discount of 0.8, and vary the number of eigenvectors to show how many PVFs are needed for good accuracy. Transferring the basis functions learned from a grid with reward in position A to a grid with the reward in position B retains 100% probability of success. We see that the transferred task matched, but can do no better than the best of the pure experiments. Results are similar with other reward functions.



(a) Reward Experiment: Number of Steps to goal. (b) Reward Experiment: Percentage of trials that reach the goal.

Figure 5: Reward Experiments on the 12x12 gridworld.

Table 1: Results comparison between experiments in which the eigenfunctions for the current grid are learned and used (pure) versus when the learned eigenfunctions for Exp 1.a are used in grids with different goal location (transfer).

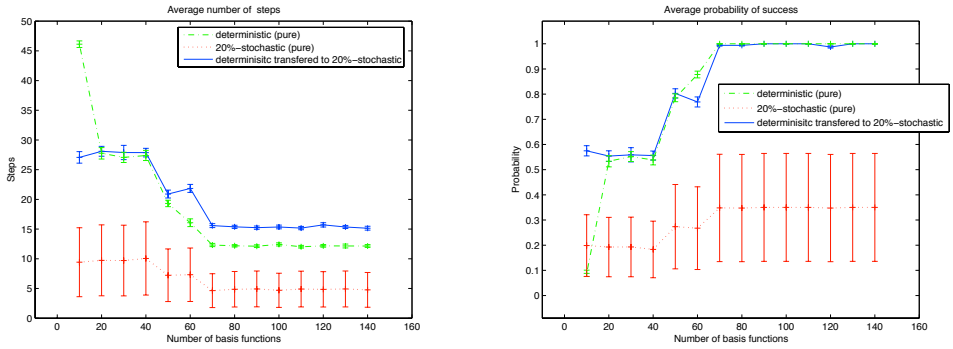
	Exp 1.a (pure)	Exp 1.b (pure)	Exp 1.b (transfer)	Exp 1.c (pure)	Exp 1.c (transfer)
Prob. of success	100%	100%	100%	100%	100%
Avg. # of steps	14.8 ± 2.1	13.6 ± 2.1	14.9 ± 3.0	7.3 ± 1.2	7.4 ± 1.2
Min/Max steps	$[5, 27]$	$[4, 22]$	$[5, 24]$	$[3, 13]$	$[2, 11]$
Avg. total discounted rew.	26.2 ± 5.6	30.0 ± 7.1	29.2 ± 8.8	53.5 ± 6.5	53.1 ± 7.3
Iterations to convergence	19	16	11	12	12

4.2 Task Transfer (Dynamics)

These *task transfer* experiments investigate transfer learning using PVFs, where the state space and basis functions are constant, but the *dynamics of the environment is varied* . More specifically, since our domains are non-deterministic, we will modify the percentage of the time that the agent will choose a random action instead of the currently learned optimal action. Since this method creates basis functions based on the actual topology of the state space, it is a natural solution to this task transfer problem.

The results in Figures 6 are for transferring the representation of two 12x12 two-room gridworlds with different dynamics. We use the 'lsqfast' algorithm in LSPI , a discount of 0.8, allow 20 iterations, and vary the number of eigenvectors. The learned policy is evaluated allowing a maximum of 50 steps, and averaged over 20 runs. Non-goal states have zero reward; the goal is consistently in position A and has reward of 100. Transferring the basis functions learned

from a 10%-stochastic grid to a 20%-stochastic grid retains 100% probability of success. We see that the transferred task matched, but can do no better than the best of the pure experiments. The 20%-stochastic pure experiment results may need to be re-run, since the results show it reaches the goal in the least number of steps, but has a poor probability of success. Results are similar when transferring between deterministic and stochastic domains.



(a) Dynamics Experiment: Number of Steps to goal.

(b) Dynamics Experiment: Percentage of trials that reach the goal.

Figure 6: Dynamics Experiments on the 12x12 gridworld.

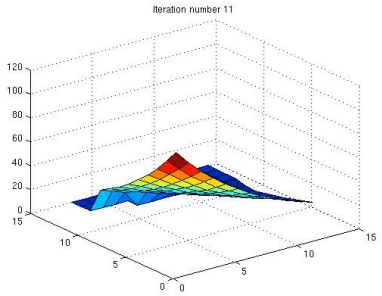
4.3 Domain Transfer (Scale)

These *domain transfer* experiments investigate transfer learning using PVFs, where the *size of the state space changes* and the the basis functions must be modified interpolated to span a larger state space. For simplicity, the reward function and state dynamics will remain constant. This is an important trype of transfer learning since the dynamics of a gridworld with no obstacles are the same regardless of scale; the basic topology is a square, the actions are North, East, South, West, and you cannot walk through the walls on the border. An agent should be able to transfer the representation it has learned in one gridworld to another. We will focus on the harder case: transferring from a smaller domain to a larger domain.

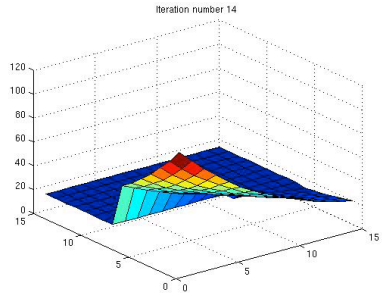
In initial work, the Nystrom extension was used to transfer the PVFs from a smaller gridworld to a larger gridworld (i.e., 10x10 grid to 20x20 grid) (Ferguson & Mahadevan, 2006). This method interpolated new PVFs for the new states in the larger domain by basically averaging the nearby known PVFs taken from the smaller domain. Proto-transfer using the Nystrom worked in several simple cases, but one downfall was that if the goal was in the section that needed interpolation, the method did not work.

Even though Table 1 shows that the extending the basis functions to larger state spaces using the Nystrom method is working well (100% for larger magnifications), the learned proto-value functions show that the interpolation may

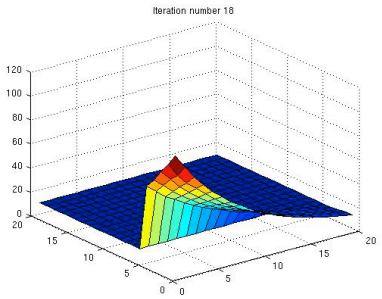
not be correct (see Figure 7). The transfer works well as long as the reward is not in the area being extended. This is another possible indication that the learned value-functions are not correct. Figure **??** shows the different results for transferring from a 10x10 gridworld to a 20x20 gridworld with the reward state in different locations. Perhaps adding states to the grid is moving the reward in a significant way and we can't deal with moving the reward and changing the state space and basis functions all at once. It is likely that the Nystrom's method of averaging is to simple for this problem and another method may be a better fit.



(a) Exp 3.a (Transfer): Learned value function when transferring from 10x10 grid to 12x12 grid.



(b) Exp 3.b (Transfer): Learned value function when transferring from 10x10 grid to 15x15 grid.



(c) Exp 3.c (Transfer): Learned value function when transferring from 10x10 grid to 20x20 grid.

Figure 7: Graphs of the value functions where the transfer learning is done using the Nystrom extension to interpolate basis function up to a larger state space, with the same reward functions.

Our method of choice, Procrustes alignment, essentially stretches the PVFs of the small domain to the size of the large domain based on several given state correspondences. In our case, we give the 4 corners of the gridworld as correspondences which an agent could easily discover. This approach easily gives 100% success when transferring from a larger domain to a smaller domain, specifically

from a 15x15 gridworld (225 states) to a 12x12 gridworld (144 states).

Our preliminary set of experiments gives the corners of a 12x12 gridworld (source) and a 15x15 gridworld (target) as the the correspondences (i.e. state 1 (source) aligns with state 1 (target), state 12 (source) aligns with state 15 (target), state 133 (source) aligns with state 211 (target), and state 144 (source) aligns with state 225 (target)). These correspondences are currently hand-coded into the system. Future work will include automatically finding these correspondences. The results of these experiments show that this technique can transfer with 100% accuracy, independent of where the reward is located.

4.4 Domain Transfer (Topology)

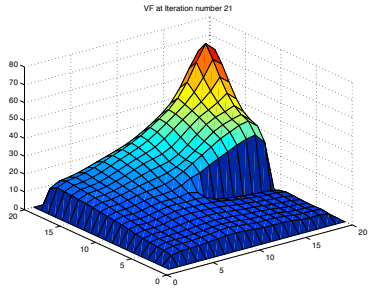
These *domain transfer* experiments investigate transfer learning using PVFs, where the *shape of the state space changes* and the the basis functions must be modified accordingly. For simplicity, the reward function and state dynamics will remain constant.

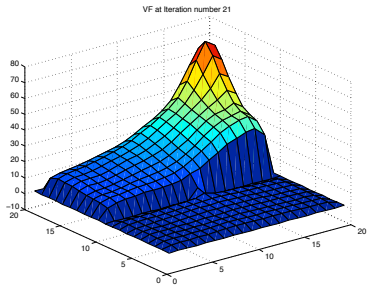
Since proto-value funcitons are constructed based on the connectedness of the states, transferring basis functions with no modification to domains of the same size but different topologies did poorly as expected. This sanity check just confirms that we will need a structured method for modifying PVFs for specific states that have changed (new obstacles, or no longer obstacles, and possibly their neighbors). Notice the difference in the purely learned value functions in Figure 8. Matrix Perturbation techniques need to be explored further before we suggest a good solution to this type of domain transfer.

5 Future Work

In this paper we have focused on proto-transfer in discrete domains only. However, PVFs also naturally abstract continuous domains (Mahadevan et al., 2006). Future work will explore proto-transfer in continuous Markov decision processes.

Using proto-value functions, an agent can transfer knowledge learned in a smaller gridworld to a larger gridworld, with high success. One exciting application for this transfer is in representation learning. Can an agent in a domain learn which domain it is in? If we have different domains as sources for transfer (i.e. a gridworld, a torroid, a chain, etc. or more abstract domains) the agent can try to transfer the knowledge from each of these domains to a knew domain (with the same reward function), and the matching domain should have a high probability of success. Therefore, which ever source domain's interpolated basis functions yield the best results on our new domain, we can conclude this new domain is of that the same type. A concrete example of when this is useful is a robot exploring the computer science department. Once it has learned the basis functions for an office and a classroom, it can use representation learning to recognize a new room as either an office or a classroom, and therefor reuse the knowledge it already has about that sub-domain.





(a) Exp 2.a (pure): Learned value function when there is a half-wall on the right side separating the two rooms which will be transfered to experiments 2.b and 2.c.

(b) Exp 2.b (pure): Learned value function when there is a full wall with a doorway separating the two rooms.

Figure 8: Graphs of the value functions where the transfer learning is done using the same reward and basis functions, but different state spaces. The eigenvectors are learned on Exp 2.a with and transferred to be used on Exp 2.b where the topology is slightly different.

References

Ferguson, K., & Mahadevan, S. (2006). Proto-transfer learning in markov decision processes using spectral methods. *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Foster, D., & Dayan, P. (2000). Using unsupervised learning methods to extract structure in value functions. *Journal of Machine Learning Research*.

Gower, J. (1975). Generalized procrustes analysis. *Psychometrika*, *40*, 33–51.

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. *Proceedings of the* 22^{nd} *International Conference on Machine Learning*.

Mahadevan, S., Maggioni, M., Ferguson, K., & Osentoski, S. (2006). Learning representation and control in continuous markov decision processes. *Proceedings of the* 21^{st} *National Conference on Artificial Intelligence*.

Sutton, R., & Barto, A. G. (1998). *An introduction to reinforcement learning.* MIT Press.

Taylor, M., Stone, P., & Liu, Y. (2005). Value functions for rl-based behavior transfer: A comparative study. *Proceedings of the* 20^{th} *National Conference on Artificial Intelligence.*

Veino, C., & Novak, M. (2003). *Tool use in juvenile rhesus macaques (macaca mulatta)* (Technical Report). University of Massachusetts, Department of Neurocience.