
Basis Function Construction for Hierarchical Reinforcement Learning

Sarah Osentoski
Sridhar Mahadevan

SOSENTOS@CS.UMASS.EDU
MAHADEVA@CS.UMASS.EDU

University of Massachusetts Amherst, 140 Governor's Drive, Amherst, MA 01003

Abstract

This paper introduces an approach to automatic basis function construction for Hierarchical Reinforcement Learning (HRL) tasks. We describe some considerations that arise when constructing basis functions for multi-level task hierarchies. We extend previous work on using Laplacian bases for value function approximation to situations where the agent is provided with a multi-level action hierarchy. We experimentally evaluate these techniques on the Taxi domain.

1. Introduction

Hierarchical reinforcement learning (HRL) approaches have been proposed to scale reinforcement learning (RL) to large domains. HRL techniques do not require the agent to reason at each time step but instead allow the agent to execute temporally-extended actions. HRL techniques decompose large problems into smaller subproblems that are simpler for the agent to solve. Several techniques, such as HAMs (Parr & Russell, 1998), options (Sutton et al., 1999), and MAXQ (Dietterich, 2000), have been introduced to explicitly represent a multi-level action hierarchy. MAXQ and hierarchical options use recursive algorithms to learn the policy on each level of the action hierarchy. The policy does not need to be defined over the entire state space; instead it can be defined on only the states at the current level of the hierarchy.

Although the subproblems are often simpler, HRL techniques can greatly benefit from generalization. HRL techniques cannot ensure that all of the subtasks will be small. Function approximation techniques will further enable HRL techniques to be scaled up to larger problems. Recently there have been several papers on automatically discovering basis func-

tions. However, most of this work has focused on automatic basis function construction for approximating value functions in situations where the agent does not have access to an action hierarchy (Keller et al., 2006; Mahadevan, 2005; Parr et al., 2007; Petrik, 2007) or where the task hierarchy is limited to a single level of abstraction (Osentoski & Mahadevan, 2007).

This paper focuses on automatic basis function construction for multi-level task hierarchies. Given a Markov Decision Process (MDP) $M = (S, A, P, R)$, and a task hierarchy H the goal is to automatically construct a low-dimensional representation Φ such that Φ provides a low-dimensional projection of the value function. The construction method should leverage H to create a compact representation Φ that respects the task hierarchy. Φ should be constructed such that the solution to M calculated using Φ closely approximates the solution of the original MDP M .

While the task hierarchy provides opportunities to speed up learning through policy reuse, value function reuse, and state abstractions, function approximation provides a powerful opportunity to create compact representations via generalization. We describe some considerations that arise when constructing basis functions for multi-level task hierarchies.

The first consideration is whether information about the reward function should be incorporated during basis function construction. Research on basis function construction has largely been divided into two categories: reward sensitive approaches (Keller et al., 2006; Parr et al., 2007; Petrik, 2007) and reward insensitive approaches (Mahadevan, 2005). Reward insensitive basis functions are an appropriate choice for low level subtasks that are often parameterized, since only one set of basis functions must be built rather than a set for each parameterization.

The second consideration is that temporal locality and spatial locality may no longer be correlated in HRL tasks. For some levels of the hierarchy, states that are sequential in the agent's decision making may no longer be close in terms of spatial locality.

The third consideration is that task hierarchies are constructed to decompose problems into simple subproblems. These subproblems allow both the policies and value functions of subtasks to be shared. Figure 1(a) shows how the Q -value function decomposes under the MAXQ framework (Dietterich, 2000) into two parts: $V_a(s)$ the expected sum of rewards obtained while executing action a and the completion function $C_i(s, a)$, the expected cumulative reward for M_i following the current policy π_i after action a is taken in state s . In order to scale, the representations created for HRL problems should decompose recursively in a similar manner. Lower level basis functions could be reused when constructing basis functions at a higher level. Figure 1(b) is a visualization of how basis functions might decompose according to the hierarchy. For a subtask i the basis functions for state s can be decomposed into two parts: $\phi_{\Omega_i}(s)$ the basis functions specific to subtask i constructed from the representation Ω_i built using the agent’s experience and $\phi_a(s)$ the basis functions from child subtasks where a is one of the child subtasks.

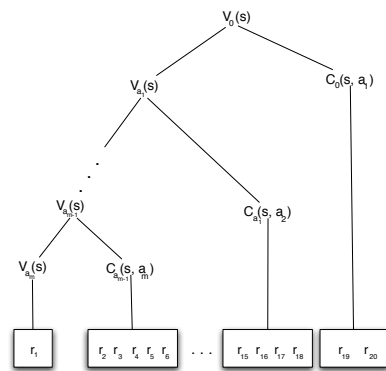
This paper describes an approach to automatically build basis functions for HRL tasks when the agent is using a pre-defined task hierarchy. We consider HRL approaches where the hierarchy constrains each macro-action to be a separate subtask. We focus on domains where state can be represented as a *factored* set of variables.

2. Hierarchical Reinforcement Learning

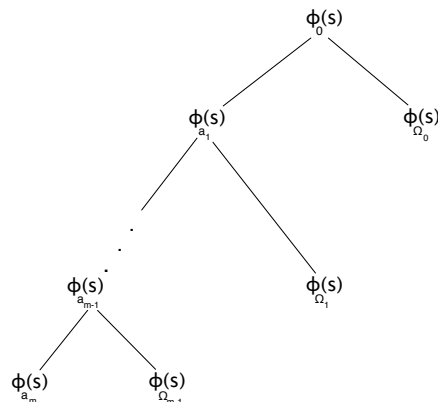
Hierarchical reinforcement learning algorithms constrain policies via a hierarchy (Barto & Mahadevan, 2003). These algorithms allow the agent to select actions that take more than one time step. Often hierarchical RL algorithms use Semi-Markov Decision Processes (SMDPs) as a model. SMDPs are a generalization of MDPs in which actions are no longer assumed to take a single time step and may have varied durations. We are specifically interested in frameworks where the agent learns at multiple levels of abstraction simultaneously. Hierarchical RL agents learn a function $Q(s, a)$ which is the expected sum of rewards for taking action a in state s , where a is either a temporally extended action or a primitive action.

2.1. Task Hierarchies for Reinforcement Learning

A task hierarchy decomposes an MDP M into a set of subtasks which can be modeled as SMDPs $\{M_0, M_1, \dots, M_n\}$ where M_0 is the root subtask which solves M . A subtask is defined to be a tuple $M_i = (\beta_i, A_i, \tilde{R}_i)$. β_i is a termination predicate. A_i is a set of actions that can be performed to achieve subtask



(a) Value function decomposition due to the task hierarchy.



(b) Representation decomposition due to the task hierarchy.

Figure 1. We explore an approach to basis function construction that exploits the value function decomposition defined by a fixed task hierarchy.

M_i . The actions can be either primitive actions from A or other subtasks. A subtask called from M_i is the child of subtask i . A subtask cannot call itself either directly or indirectly. \tilde{R}_i is a pseudo-reward function which specifies a reward function specific to M_i .

Each state s can be written as a vector of variables, X . One of the advantages of task hierarchies is that they allow abstractions to occur such that only a subset of the variables X_i are relevant to a subtask i . $X_{i,j}$ is the j th variable for subtask i . A state x_i defines a value $x_{i,j} \in \text{Dom}(X_{i,j})$ for each variable $X_{i,j}$. A task hierarchy H with an abstraction function χ is called a *state-abstracted task hierarchy*. χ_i is a function that maps a state s onto only the variables in X_i . When we refer to a state s for a specific level of the task hierarchy we are actually referring to $\chi_i(s)$.

Task hierarchies allow subtasks to be parameterized. If M_j is a parameterized subtask it is as if this task occurs many times in A_i , where M_i is the parent

task. Each parameter of M_j specifies a distinct task. β_j and \tilde{R}_j must be redefined as $\beta_j(s, p)$ and $\tilde{R}_j(s', p)$ where p is the parameter. If a subtask's parameter has many values this is the same as creating a large number of subtasks which must all be learned and creates a large number of possible actions for parent tasks.

2.2. Task Hierarchy for Taxi

We describe the task hierarchy for the Taxi task (Dietterich, 2000) pictured in Figure 2(a). The taxi task is defined as a grid of 25 states. There are four colored locations, red (R), green (G), yellow (Y), and blue (B). The task is for the agent, the taxi, to pick up the passenger located on one of the colored locations and drop the passenger at the desired destination. The factored state contains the location of the taxi, the passenger location, and the passenger destination. There are 6 primitive actions in this domain: four navigation actions, *north*, *east*, *south*, and *west* and 2 actions to access the passenger location, *pickup* and *putdown*. Each action receives a reward of -1. If the passenger is *putdown* at the intended destination a reward of +20 is given. If the taxi attempts to *pickup* a nonexistent passenger or *putdown* the passenger at the wrong destination a reward of -10 is received. If the taxi runs into the wall it remains in the same state and receives a reward of -1.

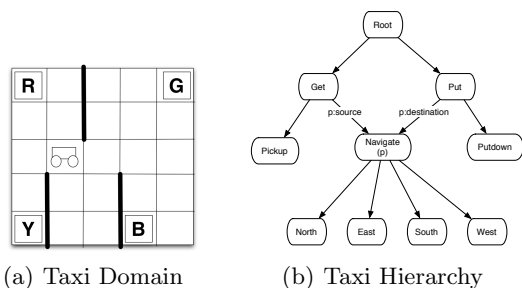


Figure 2. Taxi Task

The task hierarchy is pictured in Figure 2(b). The root node is defined over all states and state variables and can select one of two abstract actions, *get* and *put*. The *get* action can only be selected when the passenger is not located in the taxi and the *put* action can only be selected when the passenger is located in the taxi. The *get* action only considers the taxi location and the passenger location. It has access to two actions, *navigate(p)*, and *pickup*. The *put* action considers only the taxi location and the passenger destination. It has access to two actions, *navigate(p)*, and *putdown*. The *navigate* action has 4 parameter values that indicate which of the 4 locations it can navigate to and has access to the 4 navigation actions.

2.3. Solving HRL tasks

Each subtask M_i has a value function $Q_i(s, a)$ that defines the value of taking an action a in state s . $Q_i(s, a)$ is used to derive a policy π_i , typically by selecting the action with the maximum Q value for s . For this paper we focus on the MAXQ technique for HRL. However our approach easily extends to the hierarchical option framework where the hierarchies will be similar and the major difference is the update rule.

In MAXQ the value function is decomposed based upon the hierarchy. MAXQ defines Q_i recursively as:

$$Q_i(s, a) = V_a(s) + C_i(s, a) \quad \text{where}$$

$$V_i(s) = \begin{cases} \max_a Q_{i,t}(s, a) & \text{if } i \text{ composite} \\ V_i(s) & \text{if } i \text{ primitive.} \end{cases}$$

$V_a(s)$ is the expected sum of rewards obtained while executing action a . The completion function, $C_i(s, a)$, is the expected cumulative reward for M_i following the current policy π_i after action a is taken in state s . \tilde{C} is the completion function that incorporates both \tilde{R}_i and R (the real reward function) and is used only inside the subtask to calculate the optimal policy of subtask i . \tilde{Q}_i is defined as $\tilde{Q}_i(s, a) = V_a(s) + \tilde{C}_i(s, a)$. \tilde{Q} is used to select the maximum action. If \tilde{R}_i is zero, C and \tilde{C} will be identical.

When using function approximation in HRL, we assume each subtask contains a set of basis functions Φ_i and a set of k weights θ_i that are used to calculate the value function. $\phi_i(s, a)$ is a k length feature vector for state s and action a . The completion function for subtask i at time t is approximated by $\hat{C}_{i,t}(s, a) = \sum_{j=1}^k \phi_{i,j}(s, a)\theta_{i,j,t}$. The update rule for the weights is $\theta_{i,t(t+N)} = \theta_{i,t} + \alpha_i[\gamma^N(\max_{a' \in A(s')} \hat{C}_{i,t}(s', a') + V_{a',t}(s')) - \hat{C}_{i,t}(s, a)] \cdot \phi_i(s, a)$ where N is the duration of a .

3. Automatic Basis Function Construction for HRL

We focus on the graph Laplacian approach to automatic basis function construction (Mahadevan, 2005). In this approach the agent automatically constructs basis functions by first exploring the environment and collecting a set of samples. These samples are used to create a graph where the vertices are states and edges are actions. Basis functions are created by calculating the eigenvectors of the graph Laplacian.

A general overview of spectral decomposition of the Laplacian on undirected graphs can be found in (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 set of 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 $\mathcal{L}_u = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$. The set of basis functions, Φ , are the “first” k eigenvectors associated with the smallest eigenvalues of either L_u or \mathcal{L}_u . $\phi(s)$ is the *embedding* of state s defined by the eigenvectors.

3.1. Recursive Reduced Graph Approach to Basis Function Construction in HRL

One reason HRL is useful is that value functions have been shown to decompose with the hierarchy. The intuition behind our approach to basis function construction for HRL problems is that similarity should decompose much as value functions decompose. Our approach automatically constructs basis functions using basis functions from the children of a subtask. A reduced graph is created and basis function specific to the subtask are generated from this graph. Basis functions are gathered recursively from child subtasks.

The first step to our approach for representation discovery for multi-task hierarchies is to perform sample collection such that a subtask has a set of samples \mathcal{D}_i which consist of a state, action, reward, and next state, (s, a, r, s') . The agent constructs a graph from \mathcal{D}_i . The agent can leverage a state-abstracted task hierarchy by building the graph in the abstract space defined by χ_i . The graph can be built such that $\chi_i(s_1)$ is connected to $\chi_i(s_2)$ if the agent experienced a transition from $\chi_i(s_1)$ to $\chi_i(s_2)$ in \mathcal{D}_i . We call a graph over an abstract space a *state-abstracted graph*. Figure 3 describes how a state-abstracted graph can be created; this approach is similar to the approach in Osentoski and Mahadevan (2007) but uses the abstraction function χ .

State-abstracted Graph: For an MDP M with a state-abstracted task hierarchy, a state-abstracted graph G_i can be constructed for subtask i over the reduced state space defined by χ_i such that the vertices V correspond to the abstract states for subtask i $\chi_i(S)$ or a subset of abstract states $\chi_i(S)$. In the state-abstracted graph G_i , $v_1 = v_2$ for any pair of states s_1 and s_2 where $\chi_i(s_1) = \chi_i(s_2)$. An edge exists between u and v if there is an action that causes a transition between the corresponding abstract states.

3.1.1. BUILDING THE REDUCED GRAPH

In this section we describe how properties of the graph can be used to create abstractions through a reduced graph. Our approach to graph reduction requires that the original graph \mathcal{G}_i be an edge labeled graph. We define an edge labeled graph to be $G = (V, E, Z, W)$ where V is the set of vertices, E is the edge set, Z is a set of labels over E , and W is the

CreateBasis Algorithm(Subtask i , Samples \mathcal{D} , Number of local basis functions k_i , Initial policy π_0)

1. Sample Collection:

- (a) **Exploration:** Generate a set of samples \mathcal{D}_i , which consists of a state, action, reward, and nextstate, (s, a, r, s', N) for subtask, i according to π_0 . N is the number of steps a took to complete.
- (b) **Subsampling Step (optional):** Form a subset of samples $\mathcal{D}_i \in \mathcal{D}$ by some subsampling method.

2. Representation Learning:

- (a) **if** GraphReduction will be performed
Build an edge labeled graph $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}, \mathcal{Z}, \mathcal{W})$ from \mathcal{D}_i where \mathcal{Z} are labels over the edge set \mathcal{E} . State v_1 is connected to state v_2 if $\chi(s_1)$ and $\chi(s_2)$ are linked temporally in \mathcal{D}_i by an action a . $\mathcal{Z}(\chi_i(s_1), \chi_i(s_2)) = a$.
- (b) **else** Build an graph $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ from \mathcal{D}_i where state v_1 is connected to state v_2 if $\chi(s_1)$ and $\chi(s_2)$ are linked temporally in \mathcal{D}_i .
- (c) $G_i = \text{GraphReduction}(\mathcal{G}_i, k_i)$ as found in Figure 4.
- (d) Calculate the k_i lowest order eigenfunctions of the graph Laplacian of G_i .

Figure 3. CreateBasis Algorithm for Hierarchical Reinforcement Learning.

weight matrix. \mathcal{G}_i must be constructed such that the Z is the action a that caused the transition between v_1 and v_2 .

Reduced Graph: A reduced graph can be constructed for subtask i with a graph \mathcal{G}_i in the following manner: Two vertices v_1 and v_2 , corresponding to states, or abstract states, s_1 and s_2 respectively can be represented as the same abstract vertex \tilde{v} if the state variables for M_i can be divided into two groups X_i and Y_i such that:

- s_1 and s_2 differ only in their values of Y_i
- v_1 and v_2 are connected to the same set of vertices in the graph and the labels $z \in Z$ for those edges are the same.

v_1 and v_2 are merged into an abstract vertex \tilde{v} corresponding to the subset of state variables X_i .

A reduced graph can be built if M does not have an abstraction function χ associated with H or if the state-abstracted graph can be further compressed. If no nodes are merged the graph will be the original graph. Figure 4 contains the algorithm used to transform the state graph into the reduced graph and create basis functions from the reduced graph.

```

GraphReduction Algorithm(Original Graph  $\mathcal{G}_o, k_i$ )
Create reduced graph,  $G_i = (V, E, W)$ , from  $\mathcal{G}_o$ 
 $V = \mathcal{V}_o$ 
For all  $v_1 \in V$ 
  Loop through  $v_2 \in V$ 
     $V_1$  is the set of vertices such that
       $v' \in V_1 \implies v_1 \rightarrow v'$ 
     $V_2$  is the set of vertices such that
       $v' \in V_2 \implies v_2 \rightarrow v'$ 
    if  $V_1 = V_2$ 
      and the labels over the edges are the same
      and the factored states of  $s_1$  and  $s_2$ 
        can be split into two groups  $X_i$  and  $Y_i$ 
        such that the states only differ in their
        values of  $Y_i$ 
      then merge  $v_1$  and  $v_2$  into an abstract node  $\tilde{v}$ 
        corresponding to the state variables  $X_i$ 
Return  $G_i$ 
  
```

Figure 4. Graph Reduction Algorithm

3.1.2. GENERATING HIERARCHICAL BASIS FUNCTIONS

Basis functions are automatically constructed by first generating basis functions for the graph G_i using the spectral decomposition of the graph Laplacian. These basis functions are concatenated together along with basis functions recursively gathered from the child subtasks. This means that the basis functions are no longer guaranteed to be linearly independent. If necessary the bases can be reorthogonalized using Gram-Schmidt or QR decomposition.

This approach allows methods such as graph Laplacian basis functions to be scaled to larger domains since the reduced graph can greatly reduce the size of the eigenproblem that must be solved.

3.2. Example of Hierarchical Basis Function Construction on the Taxi Task

To illustrate our approach we return to the example of the *get* task. Figure 5 shows the state-abstracted graph of the *get* subtask. The four clusters of nodes correspond to the states for each passenger location for the task. Within each cluster the darker vertices correspond to states where the taxi is located in one of the colored gridstates. Dark edges are refer to edges caused by primitive actions, in this case the *pickup* action.

Figure 6 displays the reduced graph for the *get* task. The outer four nodes correspond to abstract nodes corresponding to states where the taxi is not in one of the colored grid locations. The four inner states correspond to the bottleneck states when the agent is in the same location as the passenger. The center state represents when the passenger has been picked up and

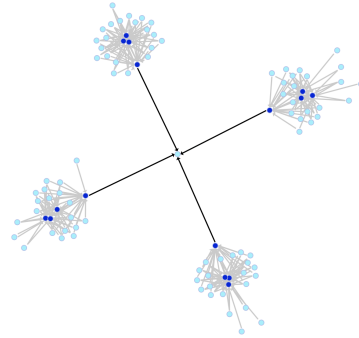


Figure 5. State-abstracted graph for the *get* subtask.

is in the taxi. Basis functions for the *get* task are constructed using eigenvectors of the reduced graph and basis functions from the navigate child subtask.

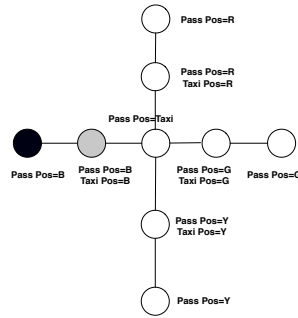


Figure 6. Reduced graph for the *get* task.

4. Experimental Analysis

We evaluated 4 different techniques: hierarchical recursive graph Laplacian basis functions, graph Laplacian basis functions using the more traditional approach, RBFs, and table-lookup on the Taxi task. The results can be seen in Figure 7. The results of each experiment was averaged over 30 trials.

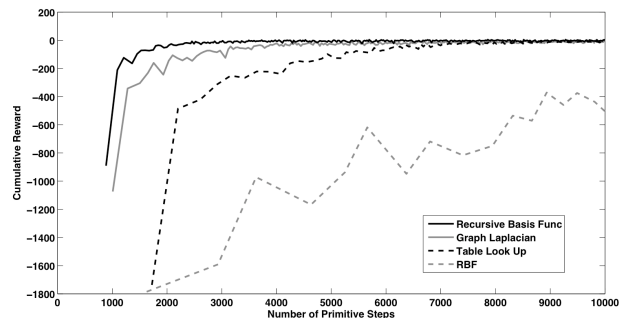


Figure 7. Results for the Taxi domain

The function approximation techniques are all using a similar number of basis functions. Our results use the normalized graph Laplacian. The recursive basis function approach used 10 local basis functions for the *navigate* subtask, 9 basis functions for *get*, and 7 basis functions for *put*. The basis functions of the graph Laplacian of the state space were created using directed graphs and the directed graph Laplacian (Johns & Mahadevan, 2007). 10 basis functions were used for all of the subtasks. It is important to note that while a similar number of basis functions were calculated for both of the graph based approaches the reduced graph is significantly smaller than the state space graph. The recursive approach also uses basis functions from lower levels in order to obtain a better approximation. The *navigate* subtask had a total of 17 basis functions created by uniformly placing the RBFs with 2 states between each RBF. The *get* and *put* subtasks had 21 basis functions created by placing the RBFs uniformly with 5 states between each RBF. We experimented with different numbers of RBFs but even doubling the number did not greatly improve performance.

5. Discussion and Future Work

This paper introduces a novel approach for basis function construction for HRL. We show that our approach enables HRL techniques to learn substantially quicker. It also allows basis function construction to be scaled to large domains where solving eigen problems is computationally prohibitive.

While we assumed that the hierarchy is given and that samples to create the basis functions are collected using the hierarchy, this is not necessary. A significant amount of work has focused on subtask creation such as (Jonsson & Barto, 2006; Mehta et al., 2008). These approaches require samples from the domain that could be used to generate basis functions for the hierarchy. Şimşek and Barto (2008) create a graph that could be modified into graphs, similar to ours, once the subgoal options are determined.

Our learned representations are useful for transfer between subtasks. Subtasks at similar levels of abstraction often have topologically similar graphs. Additionally the reduced graph provides an abstraction that should allow the representation and thus the subtask to be transferred to new learning scenarios.

Another area of future work is combining reward sensitive and insensitive approaches in HRL tasks. We have started extending BEBFs (Parr et al., 2007) to TD methods, used in HRL algorithms, using a collection of samples. However the Bellman error from TD updates is not smooth across the state space. More work is needed to extend BEBFs to TD methods or

sample based least squares approaches to SMDPs.

References

- Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Special Issue on Reinforcement Learning, Discrete Event Systems Journal*, 13, 41–77.
- Chung, F. (1997). *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society.
- Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13, 277–303.
- 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)*.
- Jonsson, A., & Barto, A. (2006). Causal graph based decomposition of factored mdps. *Journal of Machine Learning Research*, 7, 2259–2301.
- Keller, P. W., 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.
- Mehta, N., Ray, S., Tadepalli, P., & Dietterich, T. (2008). Automatic discovery and transfer of maxq hierarchies. *Proceedings of the 25th International Conference on Machine Learning* (pp. 648–655). Helsinki, Finland.
- Osentoski, S., & Mahadevan, S. (2007). Learning state-action basis functions for hierarchical mdps. *Proceedings of the 24th International Conference on Machine Learning*.
- Parr, R., Painter-Wakefield, C., Li, L., & Littman, M. (2007). Analyzing feature generation for value-function approximation. *Proceedings of the International Conference on Machine Learning (ICML)*.
- Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10* (pp. 1043–1049). MIT Press.
- Petrik, M. (2007). An analysis of laplacian methods for value function approximation in mdps. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.
- Şimşek, Ö., & Barto, A. (2008). Skill characterization based on betweenness. *Neural Information Processing Systems (NIPS)*.
- 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.