Efficient Skill Learning using Abstraction Selection

George Konidaris and **Andrew Barto**
Autonomous Learning Laboratory
Department of Computer Science
University of Massachusetts Amherst
{gdk, barto}@cs.umass.edu

Abstract

We present an algorithm for selecting an appropriate abstraction when learning a new skill. We show empirically that it can consistently select an appropriate abstraction using very little sample data, and that it significantly improves skill learning performance in a reasonably large real-valued reinforcement learning domain.

1 Introduction

Much recent research in reinforcement learning has focused on hierarchical reinforcement learning [Barto and Mahadevan, 2003], which aims to create reinforcement learning agents that use hierarchies of high level skills; one major goal of such research is the autonomous discovery of such skill hierarchies. Most existing research in skill discovery for hierarchical reinforcement learning involves small discrete state spaces, where the conjectured benefits of skill hierarchies are to focus exploration and produce transferrable skills that improve performance.

One class of reinforcement learning problems which remains difficult to solve is that of high-dimensional, continuous domains. A key approach to solving such hard problems is the use of an abstraction that reduces the number of state variables used to solve the problem. However, it is often difficult to find a single abstraction that applies to the whole problem. Although such a problem might be intrinsically high-dimensional and therefore hard to solve monolithically, it may at the same time consist of several subproblems, each of which is much easier and can be solved using only a small set of state variables. Thus, a complex human task such as driving to work, that seems infeasible to learn as a single overall problem, might be broken into a series of small subtasks (unlocking the car, starting the car, navigating to work, parking, walking inside, etc.), each of which is manageable on its own. We may therefore be able to gain an additional advantage through the use of skill hierarchies by breaking a large problem down into a series of subproblems, each of which can be solved using its own abstraction.

In this paper, we propose that if an agent has a library of abstractions available to it, then when it acquires a new skill it can select one from among them, and apply it to aid in skill learning. We present an abstraction selection algorithm, show empirically that it selects an appropriate abstraction using very little sample data, and that it significantly improves skill learning performance in the Continuous Playroom, a reasonably large real-valued reinforcement learning domain.

2 Background

2.1 Reinforcement Learning in Continuous Domains

Reinforcement learning algorithms usually (although not necessarily, e.g. policy gradient algorithms, model based methods, etc.) learn by constructing a value function V mapping states of a task to return (expected discounted future reward from a state). In many real-world applications, states are described by vectors of real-valued features, and consequently we must approximate a value function defined over real-valued inputs. This creates two problems. First, we must find a way to compactly represent a value function defined on a multi-dimensional real-valued feature space. Second, that representation must facilitate generalization.

The most common approximation scheme is *linear function approximation* [Sutton and Barto, 1998] which approximates V by a weighted sum of *basis functions* $\mathbf{\Phi}$:

$$\bar{V}(\mathbf{s}) = \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{s}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{s}), \tag{1}$$

where ϕ_i is the ith basis function. Thus learning entails obtaining a weight vector $\mathbf{w}$ such that the weighted sum in Equation 1 accurately approximates V. Since $\bar{V}$ is linear in $\mathbf{w}$, when this approximation is not degenerate there is exactly one such optimal $\mathbf{w}$; however, we may represent complex value functions this way because each basis function ϕ_i may be arbitrarily complex; in this work we use the Fourier Basis [Konidaris and Osentoski, 2008].

The most common family of reinforcement learning methods, and the methods used in this work, are *temporal difference methods* [Sutton and Barto, 1998]. Temporal difference methods learn a value function (and hence a policy) online, through direct interaction with the environment.

2.2 The Options Framework

In hierarchical reinforcement learning, apart from a given set of primitive actions an agent can also acquire and use higher-

level macro actions built out of primitive actions. In this paper we adopt the *options framework* [Sutton *et al.*, 1999], a hierarchical reinforcement learning framework that provides methods for learning and planning by adding temporally extended actions (called *options*) in the standard reinforcement learning framework. An option o consists of three functions:

$$\begin{array}{lll} \pi_o: & (s,a) & \mapsto [0,1] \\ I_o: & s & \mapsto \{0,1\} \\ \zeta_o: & s & \mapsto [0,1], \end{array}$$

where $s \in S$ is a state, $a \in A$ is an action, π_o is the *option policy* (a probability distribution over actions at each state in which o is defined), I_o is the *initiation set* indicator function, which is 1 for states where the option can be executed and 0 elsewhere, and ζ_o is the *termination condition*, giving the probability of the option terminating in each state [Sutton *et al.*, 1999]. Algorithms for creating new options must include methods for determining when to create an option or expand its initiation set, how to define its termination condition, and how to learn its policy. Policy learning is usually by an off-policy reinforcement learning algorithm so that the agent can update many options simultaneously after taking an action [Sutton *et al.*, 1998]. Creation and termination are usually performed by the identification of subgoal states, with an option created to reach a goal state and to terminate when it does so. The initiation set is then the set of states from which the goal is reachable. Previous research has selected goal states by a variety of methods, for example variable change frequency [Hengst, 2002] and local graph partitioning [Şimşek *et al.*, 2005].

3 Abstraction Selection

Humans have many sensory inputs and degrees of freedom, which viewed naively represent a very large state space. Although such a state space seems too large for feasible learning, specific sensorimotor skills almost always involve a small number of sensor features. One of the ways in which we might draw inspiration from human learning is the extent to which humans learning motor skills seem to ignore most of the sensor and motor features in their environment.

In reinforcement learning, the use of a smaller set of variables to solve a large problem is modeled using the broad notion of *abstraction* [Li *et al.*, 2006]. For this work, we consider an abstraction M_i to be a pair of functions (σ_i, τ_i), where

$$\sigma_i : S \to S'$$

is a mapping from the overall problem-specific state space S to a smaller state space S' (often simply a projection onto a subspace spanned by a subset of the variables in S), and

$$\tau_i : A \to A'$$

is a mapping from the full problem action space A to a smaller action space A' (often simply a subset of A). In addition, we assume each abstraction has an associated vector of basis functions Φ_i defined over S' using which we can define value functions.

Typically, a reinforcement learning agent tries to build a single abstraction for the entire problem; however, in the hierarchical reinforcement learning setting it may in principle try to build as many abstractions as it has skills. An agent that must solve many problems in its lifetime may thus accumulate a *library* of abstractions that it can later deploy to solve new problems.

We propose that when an agent creates a new option it should create it with an accompanying abstraction, and that if it has a library of abstractions available it can select from among them, refining the selected abstraction through experience if necessary. The agent can use the state-action samples it experiences as it creates the option to decide which of the available abstractions to use. We present an algorithm for abstraction selection in the following section.

3.1 Selecting an Abstraction

In the hierarchical reinforcement learning literature, an agent creates an option to reach a particular state (or state region) only after that region is first reached. Thus, we have a set of sample interactions that ends at the new subgoal, and we may consider it a sample trajectory for the option. Assuming the trajectory has m steps, it consists of a sequence of m state-action pairs and resulting rewards:

$$\{(\mathbf{s}_1, a_1, r_1), (\mathbf{s}_2, a_2, r_2), \ldots, (\mathbf{s}_m, a_m, r_m)\}.$$

Given a library of abstractions, we can apply each abstraction to the sample trajectory and obtain:

$$\{(\mathbf{s}_1^i, a_1^i, r_1), (\mathbf{s}_2^i, a_2^i, r_2), \ldots, (\mathbf{s}_m^i, a_m^i, r_m)\},$$

where $(\mathbf{s}_k^i, a_k^i, r_k) = (\sigma_i(\mathbf{s}_k), \tau_i(a_k), r_k)$ is a state-action-reward tuple obtained from abstraction i describing the kth state-action pair in the trajectory.

Abstraction Selection as Model Selection

If we are given the entire trajectory at once (or can store it all in memory), we may compute Monte Carlo state-value samples (s_i, R_i) for each $1 \leq i \leq m$, where $R_i = \sum_{j=i}^{m} r_j$, and the problem reduces to *model selection* [Bishop, 2006], where we select the best model (in our case, abstraction) for a regression problem. A common model selection criterion is the *Bayesian Information Criterion* (or BIC) [Schwarz, 1978], which states that:

$$\ln p(D|M_i) \approx \ln p(D|\theta_{MAP}, M_i) - \frac{1}{2}|M_i| \ln m, \qquad (2)$$

where D is the data, M_i is abstraction i, $p(D|\theta_{MAP}, M_i)$ is the likelihood of D given the maximum a priori value function parameters θ_{MAP} for abstraction i, $|M_i|$ is the number of parameters in abstraction i and m is the sample size.

BIC has two important properties. First, it controls for different sized abstractions (through the second term in Equation 2) in a principled way. Second, if we wish we can use the results of Equation 2 and Bayes Rule to obtain a probability that each abstraction is the correct one for a particular skill, naturally incorporating prior beliefs the agent may have about suitable abstractions and allowing for principled autonomous decision making.

We must now select an appropriate statistical model for the data; since we are using linear function approximation, a linear regression model is a natural choice. In such models, it is typical to assume that the target variable has mean

$\mathbf{w} \cdot \Phi_i(s)$ for state s, where $\mathbf{w}$ is a weight vector parameter and Φ_i is a set of basis functions (in our case those associated with abstraction i), and variance β^{-1}, which is assumed to be the same for all samples. However, in the case of an MDP Monte Carlo sample, all of the sample points do *not* have the same variance; a sample point's variance increases with the length of the trajectory following it. For simplicity, we model this increasing variance using a geometric series with factor $0 \leq \rho \leq 1$. Thus, sample i in a trajectory of length m is assumed to have variance $(\rho^{m-i}\beta)^{-1}$. This corresponds to a weighted least squares regression model with sample i assigned weight ρ^{m-i}. The log likelihood of this model is:

$$\ln p(D|M_i, \mathbf{w}, \beta) = -\frac{\beta}{2}e_i + \frac{m}{2}(\ln \frac{\beta}{2\pi}) + \frac{m^2 - m}{4} \ln \rho, \tag{3}$$

where β^{-1} is the variance, $\mathbf{w}$ is the function approximation weight vector, and

$$e_i = \sum_{j=1}^{m} \rho^{(m-j)}[\mathbf{w} \cdot \Phi_i(\mathbf{s}_j) - R_j]^2, \tag{4}$$

is the summed weighted squared error. Note that since maximizing Equation 3 with respect to $\mathbf{w}$ is equivalent to minimizing the (weighted) value function error (Equation 4) the resulting $\mathbf{w}$ vector can be used as an initial value function parameter.

Since we wish to do selection with very little data if possible, we must avoid overfitting. We therefore regularize by assuming a zero-mean Gaussian prior with precision α for each element of $\mathbf{w}$ (this can be discarded if desired by setting $\alpha = 0$, which results in no regularization). The maximum a posteriori (MAP) parameters are then:

$$\begin{aligned} \beta &= e_i^{-1} m \\ \text{and} \quad \mathbf{w} &= (A + \eta I)^{-1} b, \end{aligned} \tag{5}$$

where

$$\begin{aligned} \mathbf{A} &= \textstyle\sum_{j=1}^{m} \rho^{(m-j)} \Phi_i(\mathbf{s}_j) \Phi_i^T(\mathbf{s}_j) \\ \text{and} \quad \mathbf{b} &= \textstyle\sum_{j=1}^{m} \rho^{(m-j)} R_j \Phi_i(\mathbf{s}_j), \end{aligned} \tag{6}$$

with $\eta = \alpha(\beta^{-1})$. We can also rewrite e_i as:

$$e_i = \mathbf{w}^T \mathbf{A} \mathbf{w} - 2\mathbf{w} \cdot \mathbf{b} + R_c, \tag{7}$$

with $\mathbf{A}$ and $\mathbf{b}$ defined as before, and $R_c = \sum_{j=1}^{m} \rho^{(m-j)} R_j^2$.

Incremental Abstraction Selection

Thus far we have assumed that we are given the entire trajectory at once (or can store it all in memory), and can thereby compute the Monte Carlo return R_i for each sample (s_i, r_i). However, this is neither desirable nor necessary. From Equations 5 and 7, we see that in order to do selection incrementally, we are only required to compute $\mathbf{A}$, $\mathbf{b}$ and R_c incrementally in order to solve for the model parameters $\mathbf{w}$ and β and weighted error e_i, and thereby compute the BIC approximation to log likelihood (Equation 2, via Equation 3).

Following Boyan [1999], we can accomplish this by the incremental (least-squares weighted TD(1)) algorithm given in Figure 1. The algorithm is run simultaneously for each abstraction while the agent is interacting with the environment, and then when the agent creates an option the algorithm computes the associated log likelihood for each abstraction in one step. The agent then selects the abstraction with the highest log likelihood.

function Sensorimotor Abstraction Fit (i, ρ, η) :

1. **Initialization:**
   Set $\mathbf{A_0}, \mathbf{b_0}, \mathbf{z_0}, R_c$ and R_z to 0, g to 1

2. **Iteratively handle incoming samples:**
   for each incoming sample $(\mathbf{s}_t, a_t, r_t)$:

   $\mathbf{A_t} = \rho \mathbf{A_{t-1}} + \Phi_i(\mathbf{s}_t)\Phi_i^T(\mathbf{s}_t)$
   $\mathbf{b_t} = \rho \mathbf{b_{t-1}} + \rho r_t \mathbf{z_{t-1}} + r_t \Phi_i(\mathbf{s}_t)$
   $\mathbf{z_t} = \rho \mathbf{z_{t-1}} + \Phi_i(\mathbf{s}_t)$

   $R_c = \rho R_c + g r_t^2 + \rho r_t R_z$
   $R_z = \rho R_z + 2 g r_t$
   $g = \rho g + 1$

3. **Compute weights, error and variance:**
   (after m samples)
   $\mathbf{w} = (\mathbf{A_m} + \eta \mathbf{I})^{-1} \mathbf{b_m}$
   $e = \mathbf{w}^T \mathbf{A_m} \mathbf{w} - 2\mathbf{w} \cdot \mathbf{b_m} + R_c$
   $\beta = \frac{m}{e}$

4. **Compute log likelihood and BIC:**
   (quantities constant across abstractions ignored)
   $ll = -\frac{\beta}{2} e + \frac{m}{2} \ln \beta$
   return ll - $\frac{1}{2}|\Phi_i| \ln m$

Figure 1: An incremental algorithm for computing the BIC value of an abstraction i, using weight factor ρ and regularization parameter η, given a successful sample trajectory.

More than one sample trajectory may be available, or may be required to produce robust selection. Given p samples, the algorithm can be modified to run steps 1 and 2 (initialization and incoming sample processing) separately for each sample trajectory, and then sum the resulting $\mathbf{A}$, $\mathbf{b}$ and R_c variables. Steps 3 and 4 (computing the parameters and returning the BIC measure) using the summed variables would then perform a fit over all p trajectories simultaneously.

The algorithm uses $O(q_i^2)$ memory, $O(q_i^2)$ time at each step and $O(q_i^3)$ time at selection for every sensorimotor abstraction i using a function approximator with q_i features. Since we most likely wish for selection to be fast relative to learning, we can use a function approximator of the same type as we use for learning but with fewer terms for selection, and then upgrade it for learning. This additionally reduces the sample complexity for successful selection, as we show in the following section.

Once an abstraction has been selected, its weight vector $\mathbf{w}$ is a vector of value function parameters obtained by a weighted fit of the sample trajectory, and is thus hopefully a good initial value function (and hence initial policy) from which to begin learning.

4 Experiments

4.1 The Continuous Playroom

The Continuous Playroom is a real-valued version of the Playroom domain [Singh *et al.*, 2004]. It consists of an agent with three effectors (an eye, a marker, and a hand), five objects (a red button, a green button, a light switch, a bell, a ball, and a monkey), and two environmental variables (whether the light is on, and whether the music is on). The agent is in 1x1 room, and may move any of its effectors 0.05 units in one of the usual four directions. When both its eye and hand are over an object it may additionally interact with it, but only if the light is on (unless the object is the light switch). Interacting with the green button switches the music on, while the red button switches the music off. The light switches toggles the light. Finally, if the agent interacts with the ball and its marker is over the bell, then the ball hits the bell. Hitting the bell frightens the monkey if the light is on and the music is on, and causes it to squeak, whereupon the agent receives a reward of $100,000$ and the episode ends. All other actions cause the agent to receive a reward of -1. At the beginning of each episode the objects are arranged randomly in the room so that they do not overlap.

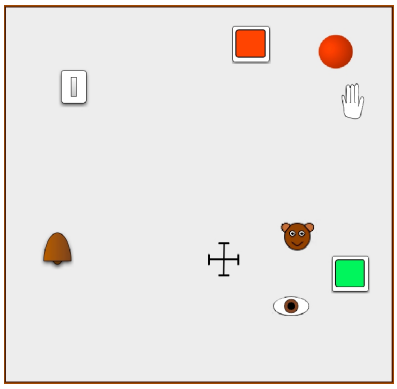

Figure 2: An example Continuous Playroom.

The agent has 13 possible actions (3 effectors each with 4 actions, plus the interact action), and a full description of the Playroom requires 18 state variables—x and y pairs for three effectors and five objects (since we may omit the position of the monkey) plus a variable each for the light and the music. Since the domain is randomly re-arranged at the beginning of each episode, the agent must learn the relationships between its effectors and each object, rather than simply the absolute location for its effectors. Moreover, the settings of the light and music are crucial for decision making and must be used in conjunction with object and effector positions. Thus, for task learning we use 120 state variables—for each of the four settings of the lights and music we use a set of 30 variables representing the difference between each combination of object and effector (Δx and Δy for each object-effector pair, so 5 objects $\times 3$ effectors $\times 2$ differences $= 30$). These features are used in an $O(3)$ independent Fourier Basis (resulting in 480 basis functions per action), and learning is performed using Sarsa(λ) ($\alpha = 0.0025$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$).

The Continuous Playroom is a good example of a domain that should be easy—and appears to humans as easy—but is made difficult by the large number of variables and interactions between variables (e.g., between Δx and Δy values for an object-effector pair) that cannot all be included in the overall task function approximator (an $O(1)$ Fourier Basis over 120 variables that does not treat each variable as independent has 2^{120} features). Thus, it is a domain in which options can greatly improve performance, but only if those options are themselves feasible to learn.

Originally, the playroom domain included options for moving each effector over each object [Singh *et al.*, 2004]. In this paper, we assume that these options must be learned efficiently, using only primitive actions, and that some given option discovery method creates them for us when the agent first encounters their goal state.

Abstraction Library and Option Learning

Since we know that the domain consists of objects and effectors, we in addition form an abstraction library of 15 abstractions, one for each combination of object and effector. Each abstraction consists of just two variables: Δx and Δy for the object-effector pair. For learning we use a full $O(7)$ Fourier Basis (resulting in 64 basis functions per action) and Sarsa(λ) ($\alpha = 0.0025$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$). For option learning without an abstraction, we discard the lights and music variables and learn using the 30 difference variables, using an $O(7)$ independent Fourier Basis (resulting in 240 basis functions per action) and Sarsa(λ) ($\alpha = 0.001$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$). The learning rate and function approximation size parameters were chosen for best performance in each case. We use $\rho = 0.99$ and $\eta = 0$ for all experiments.

4.2 Results

The first two immediate questions that abstraction selection raises are: *does the quality of trajectory data matter?*, and *how much sample data is required for selection to be accurate?* Figure 3 shows performance curves (accuracy vs. number of sample trajectories, averaged over 100 runs) for sample trajectories obtained by an optimal hand-coded option (solid lines) and by selecting actions randomly (dashed lines). Each curve color corresponds to a Fourier Basis function approximator of different order. We chose random and optimal trajectories because learning behavior (at least in the first few episodes) will lie somewhere between these two, starting off as near-random and moving towards optimal.

Figure 3 shows that both the quality of the trajectory data and the function approximator size affect selection accuracy. Better trajectory data leads to more accurate selection, although the discrepancy is not dramatic. Lower order approximators (those with fewer terms) require fewer trajectory samples to make an accurate selection, which is unsurprising since they have fewer free parameters and thus require less data to obtain a good fit. For an $O(2)$ basis, approximately 9 trajectory samples are required in this domain to select the appropriate abstraction perfectly.

The next important question is: *how much of a performance improvement does abstraction selection produce?* Figure 4 shows a learning curve (averaged over 100 runs) for

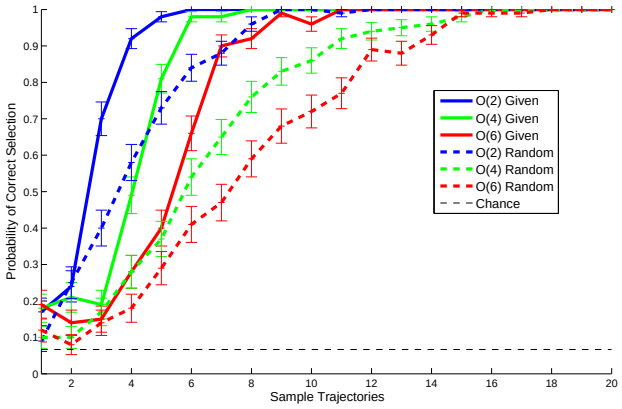

Figure 3: Selection accuracy for different orders of function approximators, given varying numbers of sample trajectories. Sample trajectories are either generated by a hand-coded optimal option policy (solid lines), or a random policy (dashed).

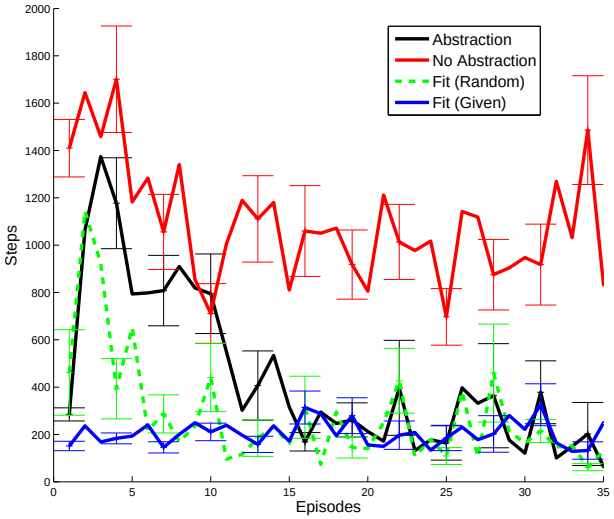

Figure 4: Learning curves for an option with and without the use of an abstraction, and with an abstraction using the initial value function fit obtained during random- and optimal-trajectory samples and abstraction selection. A random policy requires 2200 steps on average.

agents learning an option to place an effector over a target using all of the state variables compared to using the appropriate abstraction. Agents that learn using an abstraction start better, and are able to obtain better overall solutions. In addition, Figure 4 shows a learning curve for agents that start off using the fit obtained from selection with 12 sample trajectories (from either random or optimal action selection) and an $O(2)$ Fourier Basis, upgraded to $O(7)$ for learning. These agents start out better than learning from scratch, demonstrating the benefits of the initial value function obtained by selection. However, the quality of the trajectory data used for the fit significantly impacts the resulting policy, with policies obtained from fitting optimal sample trajectories performing much better than those obtained from random sample trajectories.

5 Related Work

Existing reinforcement learning research on state space reduction takes one of two broad approaches. In *state abstraction* (surveyed by Li, Walsh and Littman [2006]), a large state space is compressed by performing variable removal or state aggregation while approximately preserving some desirable property. However, without further information about the state space we cannot examine the effects of abstraction on the properties we are interested in—values or policies—without an existing value function, and so these methods have high computational and sample complexity.

The major alternative approach is to initially assume that *no* states or state variables are relevant, and then introduce *perceptual distinctions* [McCallum, 1996] by including them when it becomes evident that they are necessary for learning the skill. This requires a significant amount of data and computation to determine which variable to introduce, and then introduces them one at a time, which may require too much experience to be practical for large continuous state spaces.

Rather than starting with all features and removing some, or starting with none and adding one at a time, our method starts with a fixed number of potential abstractions and selects one prior to learning. By thus bootstrapping learning, the agent may be able to solve much larger problems, while at the same time allowing for each skill to refine its abstraction further if necessary. In addition, abstraction selection allows the agent to leverage existing abstractions it has been given or has already learned.

State abstraction and perceptual distinction methods are usually applied monolithically to a single large problems. However, some hierarchical reinforcement learning methods make use of skill-specific abstractions. Most prominently, the MAXQ hierarchical reinforcement learning formalism [Dietterich, 2000] assumes a hand-designed abstraction for each level in the skill hierarchy. On the other end of the spectrum, Jonsson and Barto [2001] have shown that each option can learn its own abstraction from scratch in discrete domains using a perceptual distinction method. However, in general hierarchical reinforcement learning methods with abstraction have been used only in relatively small discrete domains.

Another related method is work by van Seijen et al. [2007], where an agent with multiple representations is augmented with actions to switch between them. This fits neatly into the reinforcement learning framework, but does not appear likely to scale up to large numbers of representations since each new representation adds a new action to every state.

6 Discussion and Conclusion

The benefits of acquired skill hierarchies have been well examined in the context of small discrete domains. However, in high-dimensional continuous domains there may be additional benefits. Konidaris and Barto [2008] have shown that

skill discovery in continuous domains improves performance, and suggest that it does so because the agent employs a distinct function approximator to each subskill and can thereby obtain a better overall solution than it could have using a single monolithic function approximator.

Abstraction selection opens up a further advantage to skill acquisition in high-dimensional continuous domains, allowing an agent to exploit abstractions it has been given or has already learned. In an environment where an agent may acquire many skills over its lifetime this may represent a great potential efficiency improvement, that in conjunction with a good skill acquisition algorithm could enable reinforcement learning agents to scale up to higher dimensional domains. Additionally, abstraction selection opens up the possibility of *abstraction transfer*, where an agent that has learned a set of skills may benefit from the abstractions refined for each, even if it never uses those skills again.

Although we only directly consider once-off selections in this paper, an agent may continue to run the abstraction selection once an initial selection has been made, allowing it to modify its choice later (when it has more sample trajectories). We expect that in many scenarios an agent will select an abstraction early and then repeatedly reconsider its selection if its selection metrics for that skill later change significantly.

Our experiments show that abstraction selection along with a library of available abstractions can be used to make skill learning more efficient in a relatively high-dimensional continuous problem. In very high-dimensional problems, we expect that both temporal and state abstraction methods will be required to create effective learning agents. Abstraction selection provides a useful common ground where both types of abstraction can be used together to improve performance.

Acknowledgements

We would like to thank Scott Kuindersma, Bruno Ribeiro, Sarah Osentoski, Lihong Li and our anonymous reviewers for their useful feedback. Andrew Barto was supported by the Air Force Office of Scientific Research under grant FA9550-08-1-0418.

References

[Barto and Mahadevan, 2003] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems*, 13:41–77, 2003. Special Issue on Reinforcement Learning.

[Bishop, 2006] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York NY, 2006.

[Boyan, 1999] J.A. Boyan. Least squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pages 49–56, 1999.

[Dietterich, 2000] T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Hengst, 2002] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250, 2002.

[Jonsson and Barto, 2001] A. Jonsson and A.G. Barto. Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, pages 1054–1060, 2001.

[Konidaris and Barto, 2008] G.D. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains. Technical Report UM-CS-2008-24, Department of Computer Science, University of Massachusetts Amherst, July 2008.

[Konidaris and Osentoski, 2008] G.D. Konidaris and S. Osentoski. Value function approximation in reinforcement learning using the Fourier basis. Technical Report UM-CS-2008-19, Department of Computer Science, University of Massachusetts, Amherst, June 2008.

[Li *et al.*, 2006] L. Li, T.J. Walsh, and M.L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[McCallum, 1996] A. McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996.

[Schwarz, 1978] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[Şimşek *et al.*, 2005] Ö. Şimşek, A.P. Wolfe, and A.G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005.

[Singh *et al.*, 2004] S. Singh, A.G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Proceedings of the 18th Annual Conference on Neural Information Processing Systems*, 2004.

[Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 1998] R.S. Sutton, D. Precup, and S.P. Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 556–564, 1998.

[Sutton *et al.*, 1999] R.S. Sutton, D. Precup, and S.P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[van Seijen *et al.*, 2007] H.H. van Seijen, B. Bakker, and L.J.H.M. Kester. Reinforcement learning with multiple, qualitatively different state representations. In *Proceedings of NIPS 2007 Workshop on Hierarchical Organization of Behavior*, 2007.