Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining

George Konidaris, Andrew G. Barto

Technical Report UM-CS-2008-24

Autonomous Learning Laboratory
Computer Science Department
University of Massachusetts Amherst
{gdk, barto}@cs.umass.edu

Abstract

We introduce skill chaining, a skill discovery method for reinforcement learning agents in continuous domains. Skill chaining produces chains of skills leading to an end-of-task reward. We demonstrate experimentally that skill chaining is able to create appropriate skills in a challenging continuous domain and that doing so results in performance gains.

1 Introduction

Much recent research in reinforcement learning has focused on hierarchical reinforcement learning methods [1] and in particular the *options framework* [2], which adds principled methods for planning and learning using high level skills (called options) to the standard reinforcement learning framework.

An important research goal is the development of methods by which an agent can discover new skills autonomously, and thereby build its own high-level skill hierarchies. Several methods exist for discovering new options in discrete domains; however, none are immediately extensible to or have been successfully applied in continuous domains.

We introduce skill chaining, a skill discovery method for reinforcement learning agents in continuous domains. Skill chaining produces chains of lightweight skills leading to a salient event—where salience can be defined simply as an end-of-task reward, or as a more sophisticated heuristic (e.g., an intrinsically interesting event [3, 4]). The goal of each skill in the chain is to reach a state where its successor skill can be executed. We demonstrate experimentally that skill chaining is able to create appropriate skills in the Pinball domain, and that doing so results in performance benefits.

2 Hierarchical Reinforcement Learning and the Options Framework

An option o is a closed loop control policy unit consisting of three components [2]: an *option policy* π_o mapping state-action pairs over which the option is defined to an execution probability; an *initiation set* indicator function I_o, which is 1 for states where the option can be executed and 0 elsewhere; and a *termination condition* β_o, giving the probability of the option terminating at each state in which it is defined.

The options framework provides methods for learning and planning using options as temporally extended actions in the standard reinforcement learning framework [5]. Moreover, an option policy can be learned as just another reinforcement learning problem, given a reward function for the option. Thus, a reinforcement learning agent can learn its own options.

Methods for creating new options must include a method for determining when to create an option (skill discovery) or expand its initiation set, how to define its termination condition, and how to learn its policy. Policy learning is usually performed by an off-policy reinforcement learning algorithm so that the agent can update many options simultaneously after taking an action [6]. Creation and termination are typically performed by the identification of goal states, with an option created to reach a goal state and terminate when it does so. The initiation set is then the set of states from which the goal is reachable.

In previous research, goal states have been selected by a variety of methods. The most common skill discovery approach is to compute visit or reward statistics over individual states to identify useful subgoals [7, 8, 9]. Graph-based methods [10, 11, 12] build a state graph and use its properties (e.g., local graph cuts [12]) to identify subgoals. In domains with factored state spaces, the agent may create options that aim to change infrequently changing variables [13, 14]. Finally, some methods extract options by exploiting commonalities in collections of policies over a single state space [15, 16, 17, 18].

All of these methods compute some statistic over individual states, in graphs derived from a set of state transitions, or rely on each state variable having finitely many values. These properties are unlikely to easily generalize to continuous spaces, where an agent may never see the same state twice.

In discrete domains, the primary reason for creation an option to reach a subgoal state is to make that state *prominent* in learning; a state that may once have been difficult to reach can now be reached by the agent using a single decision (to invoke the option). This has the effect of restructuring the connectivity of the MDP from the agent's point of view, so that the subgoal is now connected to every state in its initiation set. Another potential reason is transfer: if options are learned in an appropriate space they can be used in later tasks to speed up learning. If the agent faces a sequence of tasks in the same state space, then options learned in it are portable [15, 16, 17, 18]; if it faces a sequence of tasks in different but related state spaces, then the options must be learned in a space common to all the tasks [19].

In continuous domains, there is a further reason for creating new options. An agent using a fixed basis set to solve the underlying problem must necessarily obtain an approximate solution. Creating new options with their own function approximators concentrated in interesting parts of the state space may result in better overall policies, freeing the primary value function from having to represent the complexities of the individual subpolicies.

3 Option Creation in Continuous Domains

Several difficulties that are present but less apparent in discrete domains become more immediate when considering creating new skills in continuous domains.

Most importantly, all existing skill discovery identify a single goal state as an option target. In continuous domains where the agent may never see the same state twice, this must be generalized to a *target region*. However, simply defining the target region as a local neighbourhood around a point will not necessarily capture the goal of a skill. For example, many of the methods outlined above target states that are difficult to get to—a too-small neighbourhood may make the target region too difficult to reach; conversely, a too-large neighbourhood may include regions of the state space that are not difficult to reach at all. Similarly, we cannot easily compute statistics over state regions or over state-region connectivity graphs when defining the size and shape of the target region is part of the problem.

One useful heuristic for skill discovery is that useful skills are highly likely to lie on a solution path, because otherwise they are unlikely to be useful in solving the overall problem. If the agent is facing a sequence of problems, then the option should be likely to lie on the solution path of some of those problems (although not necessarily the one the agent is currently solving).

While in discrete domains it is common for an option's initiation set to expand arbitrarily as the agent learns a policy for successfully executing the option, in continuous spaces this is not desirable. In discrete domains without function approximation an exact policy to reach a subgoal can always be represented; in continuous domains with function approximation (or even discrete domains with function approximation), it may only be possible to consistently represent such a policy locally. An option in a continuous domain should be able to very consistently solve a simpler problem than the overall task using a much simpler policy that will probably be only locally valid.

Similarly, an option policy in a continuous domain should be simpler to represent than the overall task policy. A value table in a discrete domain is a relatively simple data structure, and updates to it take constant time. In a continuous domains with many variables, value function approximation may require hundreds of even thousands of basis functions to represent the overall task's value function and updates are at least linear time. Therefore, "lightweight" options (that use very few basis functions relative to the number used to solve the overall problem) are desirable in high-dimensional domains, or in domains where we may wish to create many skills.

4 Skill Chaining

Since a useful option lies on the solution path, it seems intuitively obvious that the first option we create should have the aim of consistently reaching the goal. The high likelihood that the option can only do so from a local neighbourhood around the goal suggests a follow-on step: we could create an option whose goal is to *reach a state where the first option can be executed.*

In this section we formally define skill chaining, a method which formalizes this intuition to create chains of skills to reach a salient event. First, we describe how to create an option given a *salience trigger function* that it is the option's goal to trigger.

4.1 Creating a Skill to Trigger a Salient Event

Given an episodic task defined over state space S with reward function R, we define a salience trigger function f over S that is 1 on states designated salient, and 0 everywhere else. We now wish to create an option o_f, with the goal of reaching a state $s \in S$ where $f(s) = 1$. To do so, we must define o_f's termination condition, reward function and initiation set.

Since o_f's goal is to reach a state s where $f(s) = 1$, we can use f directly as o_f's termination condition. We can also use the task's reward function R directly as o_f's reward function since we aim for a policy that reaches f with the highest return in the overall task, except that we add in an option completion reward as appropriate.

Given these, we can use a standard reinforcement learning algorithm to learn o_f's policy. If we would like o_f to be "lightweight" relative to the overall problem, we allocate it fewer basis functions with which to learn that policy than are being used to solve the overall task.

Obtaining o_f's initiation set is a harder problem because it should represent the likelihood of o_f's policy reaching f, and o_f's policy must be learned. We can treat this as a standard classification problem, by keeping track of states in which o_f has been executed and succeeded and in which it has been executed and failed, and using these as training examples. A classification algorithm that handles a potentially nonstationary classification problem over continuous feature vectors can then be employed to learn o_f's initiation set.

4.2 Chaining Skills by Propagating Salience

Given a *root event* r, which for the purposes of this discussion we consider to indicate terminal states of the underlying task (i.e., r is 1 for terminal states and 0 elsewhere), we create a chain of skills as follows. First, the agent creates skill o_r to reach r and proceeds to learn a good policy for that skill, as well obtaining a good estimate of its initiation set. The initiation set delineates the state space region in which the option can consistently be successfully executed; this will probably be only a small portion of the state space and so we consider each option to be *short-range*.

Once that has been done, we may then propagate the salience of the root event by designating o_r's initiation set as a salient event trigger. This causes the creation of a new option o_c to reach the initiation set of o_r; o_c is defined as discussed above.

This procedure is then repeated, eventually resulting in a chain of short-range skills leading from any state in which the agent may start to the termination region. This process is depicted in Figure 1.

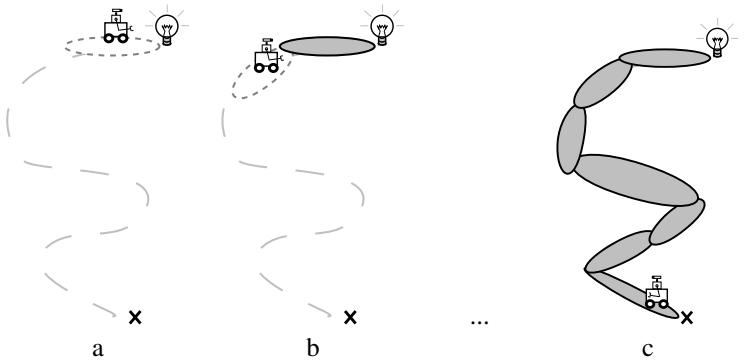

Figure 1: An agent creates skills using skill chaining. First, the agent encounters a salient event and creates a skill to reach it (a). Entering the first skill then becomes a salient event, which later triggers the creation of a second skill to reach the first (b). Finally, after many trajectories the agent has created a chain of skills to reach the original salient event (c).

Note that although we have assumed the task's end-of-episode indicator function as the root salient event, we are free to use *any* set of salient event triggers as root events. We may thus include measures of novelty or other intrinsically motivating events [3, 4] as triggers, heuristic triggers, events that are interesting for domain-specific reasons (e.g., physically meaningful events for a robot).

5 The Pinball Domain

The experiments in this paper use the Pinball domain, depicted in Figure 2. The agent's goal is to maneuver the blue ball (in the lower left corner) in the hole in the upper right. The ball's state is describe by four variables: x, y, $\dot{x}$ and $\dot{y}$, and it moves under a drag coefficient of 0.995. Since collisions with obstacles are elastic, the agent may either avoid or make use of the obstacles between it and the hole, using one of five actions: adding or subtracting a small amount of force to $\dot{x}$ or $\dot{y}$ (which incurs a penalty of 10), or leaving them unchanged (which incurs a penalty of 1). The agent receives a reward of $10,000$ points for reaching the hole, which it must do within $100,000$ steps.

The Pinball domain is an appropriate continuous domain for skill discovery because the

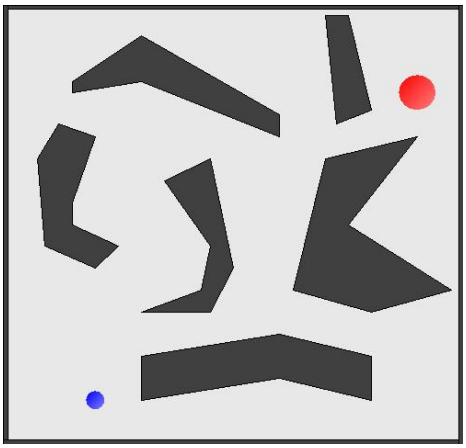

Figure 2: The Pinball Domain

dynamic aspects of the ball and the sharp discontinuities in the domain make it difficult for control and for function approximation. It seems clear that while a solution with a flat learning system is possible, there is space for acquired skills that could result in a better solution.

To learn to solve the overall task, we used Sarsa[1] ($\alpha = 10^{-4}, \gamma = 1, \epsilon = 0$) with linear function approximation, employing a weakly coupled Fourier Basis [20], using 7 basis functions per variable and up to order 2 paired variable basis functions (resulting in 44 basis functions per action).

For each learned option we employed Q-learning ($\alpha = 10^{-3}, \gamma = 1, \epsilon = 0$) with the same function approximator. An option could be executed when the agent was inside its initiation set and outside of the option's target region. Its initiation set was learned by a nearest-neighbor classifier, with each option execution allowed 200 steps to reach the goal before being labelled unsuccessful, whereupon the state from which the option execution started was used as a negative example; the starting, ending, and last fifteen states in a successful trajectory were used as positive examples. Off-policy updates to the option for states outside its initiation set were discarded, as were updates resulting from unsuccessful trajectories.

A newly created option was first allowed a "gestation period" of 10 episodes where it could not be executed and its policy was updated using only off-policy learning. This allowed the option to have a reasonable starting policy from which its initiation set could be learned. Once the option was made available to the agent, the agent waited 10 episodes to get a good estimate of the option's initiation set. Thereafter, entering the option's initiation set was designated as salient, resulting in the creation of another new option.

[1] Exploratory experiments with Sarsa(λ) with a non-zero λ did not yield a performance benefit, and in some cases led to degraded performance.

Creating a new option requires expanding the overall task action-value function. We initialized the new action's Q values to the Q values of the primitive action corresponding to not changing the ball's velocity plus a boost term (100 for the root option, reduced by 20% for each following option). These values were selected to encourage the agent to execute newly created options without giving them so high an initial value that it would do so indefinitely even if they did not perform well.

These methods were chosen because they were the simplest ones that worked; however, in an application more sophisticated measures of option readiness and value function initialization would be desirable.

6 Results

Figure 3 shows the performance (averaged over 100 runs) in the Pinball domain for agents employing skill chaining, agents starting with pre-learned options, and agents without options. The agents with pre-learned options acquired them during 200 episodes of solving the Pinball task, whereupon their task value functions were reset and the learned options were frozen. The graph shows that the agents employing skill chaining initially do about as well as agents without options but later achieve significantly better performance.

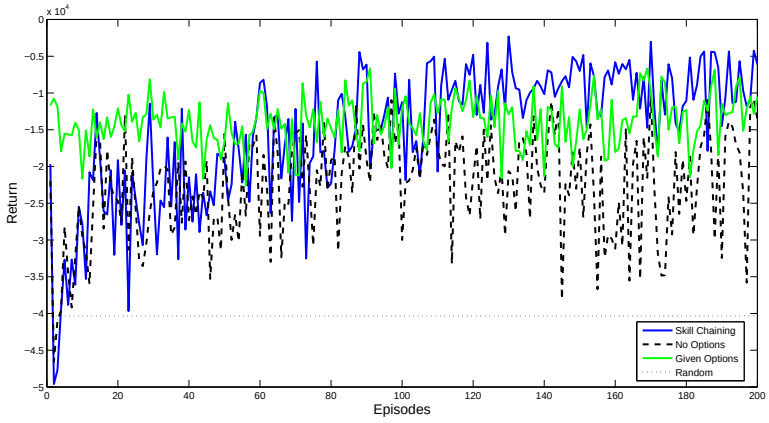

Figure 3: Performance in the Pinball domain (averaged over 100 runs) for agents employing skill chaining, agents with given options, and agents without options. The performance of an agent that chooses its actions randomly is given for reference.

Agents starting with pre-learned options do very well immediately; this shows that it is primarily having the options themselves, and not a byproduct of learning them, that leads to performance gains. However, they do not do as well by 200 episodes as the agents that learned the options themselves, suggesting that it is not *just* the

options that lead to better performance. This may be explained in part as a failure of exploration. Because the options receive a uniform termination reward for reaching any state in their target region without regard to that state's value, using the learned options exclusively can lead to a slightly worse overall solution than one that also makes use of primitive actions. The agents that learn the options themselves likely have sufficient information in their action-value functions to switch to primitive actions where necessary to make up the difference, but the agents with pre-learned options do not (since we did not perform off-policy learning for the overall task value function) because they immediately learn to execute the options in sequence and are unlikely to deviate from that solution.

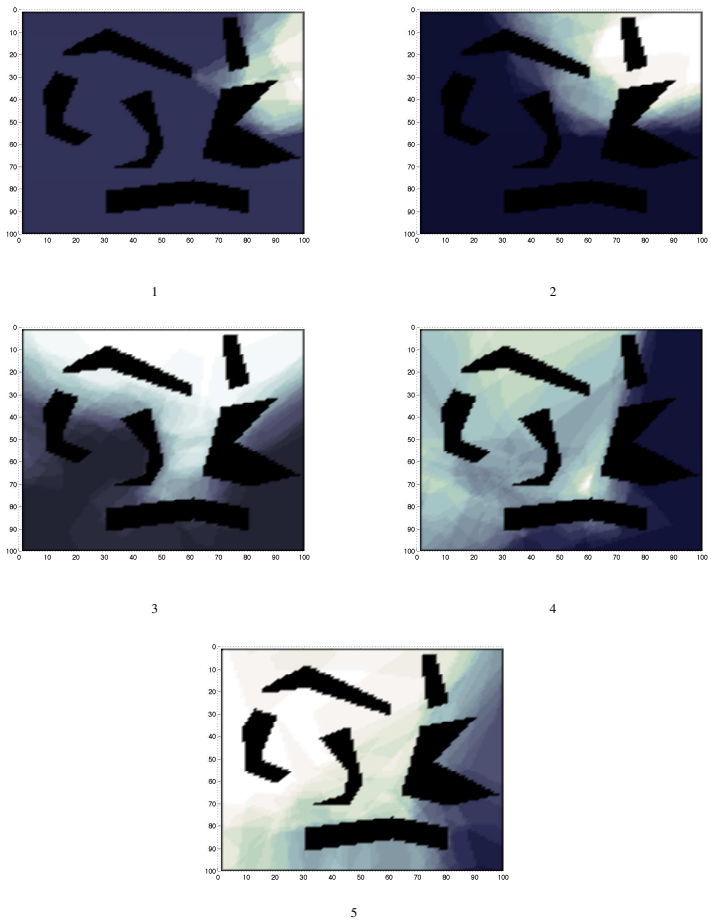

Figure 4: Initiation sets for a typical set of five skills discovered in the Pinball domains.

Figure 4 depicts the initiation sets of a typical set of five skills created in the Pinball domain. Each graph is colored according to the number of points in the initiation set

for a given (x, y) coordinate, varying $\dot{x}$ and $\dot{y}$ over the set $\{-1.0, -0.5, 0, 0.5, 1.0\}$. The initiation sets clearly show local skills specialised for different regions of the state space. Note that for the last two sets the initiation sets are overly permissive because in later episodes the agent visits some areas infrequently.

The best solution found over all skill chaining runs has return 9800, and is shown in Figure 5. The skills employed in the solution are shown in different colors.

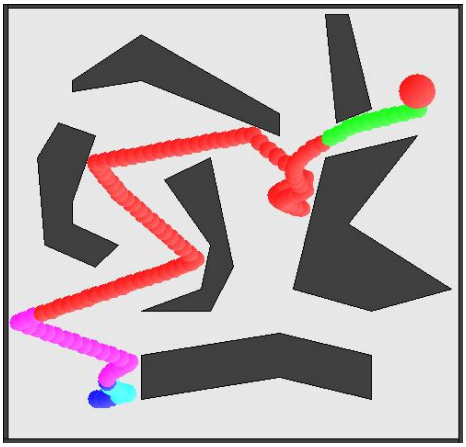

Figure 5: The best solution found for the Pinball domain.

7 Discussion

The performance gains demonstrated in the previous section show that skill chaining (at least using end-of-episode salience) does not speed up the initial performance of agents using it but improves their their asymptotic performance. Thus a likely primary benefit of skill chaining is that it reduces the representational burden of the overall value function, allowing each option to focus on representing its own local policy and thereby achieving a better overall solution. One way to achieve even better policies might be to focus the function approximator of each option around its target region. This could be achieved by implementing a non-linear warping of the state variables whereby points near the target region are projected to a larger segment of the unit interval than points farther away. We expect that this would lead to be a further improvement in both the quality and consistency of the individual solutions.

Another avenue that may lead to further performance benefits would be to include a more sophisticated notion of salience: *any* indicator function can be substituted for (or added to) the end-of-episode one used in this paper. A method that could identify regions likely to lie on the solution trajectory before a solution has been found would result in the kinds of early performance gains sometimes seen in discrete skill discovery methods.

In this paper we did not need to use a "light" function approximator to represent each option's value function, since the overall task could be solved with a small number of basis functions. However, we expect that lightweight options will be required for domains with more variables (and especially domains where those variables cannot be treated as weakly coupled). In very complex domains such as robotics, where the state space may contain hundreds or even thousands of state variables, we may require a more sophisticated approach. One such approach is *abstraction selection* [21], where the agent has a set of possible abstractions (e.g., sets of related features) from which it can select when creating a new option, based on an initial sample trajectory (as is obtained the first time an agent reaches a salient event). Each option thus starts with its own abstraction, potentially resulting in a much easier learning problem.

8 Summary

We have introduced skill chaining, a skill discovery method for We introduce skill chaining, a skill discovery method for reinforcement learning agents in continuous domains. Experiments in the Pinball domain show that skill chaining is able to discover appropriate skills and that its application results in performance gains.

References

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

[2] 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.

[3] 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.

[4] A.G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the International Conference on Developmental Learning*, 2004.

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

[6] 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.

[7] B.L. Digney. Learning hierarchical control structures for multiple tasks and changing environments. In R. Pfeifer, B. Blumberg, J. Meyer, and S.W. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland, August 1998. MIT Press.

[8] A. McGovern and A.G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368, 2001.

[9] Ö. Şimşek and A.G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758, 2004.

[10] I. Menache, S. Mannor, and N. Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, pages 295–306, 2002.

[11] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty First International Conference on Machine Learning*, pages 560–567, 2004.

[12] 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.

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

[14] A. Jonsson and A.G. Barto. A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the Twenty Second International Conference on Machine Learning*, 2005.

[15] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, 1995.

[16] D.S. Bernstein. Reusing old policies to accelerate learning on new MDPs. Technical Report UM-CS-1999-026, Department of Computer Science, University of Massachusetts at Amherst, April 1999.

[17] T.J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, Department of Computer Science, University of Massachusetts, Amherst, 1999.

[18] M. Pickett and A.G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference of Machine Learning*, pages 506–513, 2002.

[19] G.D. Konidaris and A.G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[20] 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.

[21] G.D. Konidaris and A.G. Barto. Sensorimotor abstraction selection for efficient, autonomous robot skill acquisition. In *Proceedings of the 7th IEEE International Conference on Development and Learning*, 2008. To appear.