Building Portable Options: Skill Transfer in Reinforcement Learning

George Konidaris Andrew Barto

Technical Report 2006-17

Autonomous Learning Laboratory
Computer Science Department
University of Massachusetts at Amherst

1st March 2006

Abstract

The options framework provides a method for reinforcement learning agents to build new high-level skills. However, since options are usually learned in the same state space as the problem the agent is currently solving, they cannot be ported to other similar tasks that have different state spaces. We introduce the notion of learning options in agent-space, the portion of the agent's sensation that is present and retains the same semantics across successive problem instances, rather than in problem-space. Agent-space options can be reused in later tasks that share the same agent-space but are sufficiently distinct to require different problem-spaces. We present experimental results that demonstrate the use of agent-space options in building reusable skills.

1 Introduction

A great deal of recent research in reinforcement learning has focussed on hierarchical methods (Barto & Mahadevan, 2003) and in particular the *options* framework (Sutton et al., 1999), which is a principled way of learning and using macro-actions for hierarchical learning and temporal abstraction in reinforcement learning. Unfortunately most option learning methods learn useful macro-actions within the same state space as the reinforcement learning problem the agent is solving at the time. Although this can lead to faster learning on later tasks in the same space, learned options would be more useful if they could be reused in later related but distinct tasks that may have different state spaces.

We propose a method for learning portable options by learning over two separate representations, as introduced in Konidaris and Barto (2005): a representation in *problem-space* that is Markov for the particular task at hand, and one in *agent-space* that may not be Markov but is retained across successive task instances (each of which may require a new problem-space, possibly of a different size or dimension). Agents that learn options only in agent-space obtain options that can be reused as macro-actions in future problem-spaces because the semantics of agent-space remain consistent across tasks.

We present the results of an experiment (using the Lightworld domain) that demonstrates that agent-space option learning results in options that significantly improve performance across a sequence of related but distinct tasks.

2 Background

2.1 The Options Framework

An option consists of the following three components (Sutton et al., 1999):

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

where π_o determines the *option policy* (giving a probability distribution over each state-action (s, a) pair that the option is defined in), I_o is the *initiation set*, which is 1 for the states where the option can be started from and 0 elsewhere, and β_o is the *termination condition*, which gives the probability of the option terminating in each state. The options framework (Sutton et al., 1999) provides methods for learning and planning using options as actions into the standard reinforcement learning framework (Sutton & Barto, 1998).

Methods for learning new options must include a method for determining when to create an option or expand its initiation set, how to describe its termination condition, and how to learn its option policy. Policy learning is usually performed by a standard off-policy reinforcement learning method so that the agent can update all of the options simultaneously when it takes an action (Sutton et al., 1998).

Creation and termination are usually performed by the identification of goal states, so that an option is created to reach a particular goal state and terminates when it does so. The initiation set is then expanded to all of the states from which a goal state is reachable. Previous research has selected goal states by frequency of visit and reward gradient (Digney, 1998), frequency of visit on successful trajectories (McGovern & Barto, 2001), relative novelty (Şimşek & Barto, 2004), clustering algorithms and value gradients (Mannor et al., 2004), local graph partitioning (Şimşek et al., 2005) and saliency (Barto et al., 2004; Singh et al., 2004). Other research has focussed on extracting useful options by exploiting commonalities in collections of policies over a single state space (Bernstein, 1999; Perkins & Precup, 1999; Pickett & Barto, 2002; Thrun & Schwartz, 1995).

All of these methods have been used to learn options in the same state space in which the agent is performing reinforcement learning, and thus can only be reused for the same problem or for a new problem in the same space. Additionally, the available state abstraction methods (Jonsson & Barto, 2001; Hengst, 2002; Jonsson & Barto, 2005) only allow for the automatic selection of a subset of this space for option learning. Ravindran and Barto (2002; 2003) employ an MDP minimisation framework to obtain compact representations of options (which they term *relativized options*). Relativized options are difficult to obtain because they require significant computational resources, and are only portable across tasks when the agent is able to determine an appropriate transformation from the compact option to the current state space.

2.2 Sequences of Tasks and Agent-Space

In this paper we are concerned with an agent that is required to solve a sequence of related but distinct tasks. The tasks are related in the sense that the same agent is required to solve a sequence of variations on the same type of task. We define a *sequence of related tasks* as follows.

The agent experiences a sequence of environments generated by the same generative world model (e.g., they have the same physics, the same types of objects may be present in the environments, etc.). From the sensations it receives in each environment, the agent creates two representations. The first is a state descriptor that is sufficient to distinguish Markov states in the environment. This induces a Markov Decision Process (MDP) with a fixed set of actions (because the agent does not change) but a set of states, transition probabilities and reward function that depend only on the environment the agent is currently in. The agent thus works in a different state space with its own transition probabilities and reward function for each task in the sequence. We call this state space *problem-space*.

The agent also uses a second representation from the sensations that are consistently present and retain the same semantics across tasks. This space is shared by the sequence of problems, and we call it *agent-space*. These two representations stem from two different representational requirements: problem-space models the Markov description of a particular environment, and agent-space models the (potentially non-Markov) commonalities across a set of environments. We thus term a sequence of problems *related* if it consists of environments that share an agent-space.

This approach is distinct from that taken in prior reinforcement learning research on finding useful macro-actions across sequences of tasks (Bernstein, 1999; Perkins & Precup, 1999; Pickett & Barto, 2002; Thrun & Schwartz, 1995), where the tasks must be in the same state space but may have different reward functions. An appropriate sequence of tasks under our definition requires only that the agent-space semantics remain consistent, so each task may have its own completely distinct state space. Konidaris and Barto (2005) have used the same distinction to learn shaping functions in agent-space to speed up learning across a sequence of tasks.

One simple example of an appropriate sequence of tasks would be a sequence of buildings where a robot equipped with a laser range finder is required to reach a particular location in the building. Since the laser-range finder readings are noisy and non-Markov, the robot would likely build a metric map of the building as it explores in order to localize, forming the problem space. The laser range finder readings themselves form the agent space, because their meaning is consistent across all of the buildings. The robot could eventually learn options in that space corresponding to macro-actions like moving to the nearest door. Because these options are based solely on sensations in agent space without referencing problem-space (any individual metric map), they can be used to speed up learning and exploration in any building that the robot encounters in the future.

3 Options in Agent-Space

We consider an agent solving n problems, each with its own state space, denoted $S_1, ..., S_n$, and a single action space A. We view the ith state in task S_j as consisting of the following attributes:

$$s_i^j = < d_i^j, c_i^j, r_i^j >,$$

where d_i^j is the problem-space state descriptor (sufficient to distinguish this state from the others in S_j, perhaps just i), c_i^j is an agent-space sensation, and r_i^j is the reward obtained at the state. We are not concerned here with the form of d_i^j, except to note that it may contain or be disjoint from c_j^i. The goal of reinforcement learning in each task S_j is to find a policy π_j that maps each state to an action in A so as to maximise return:

$$\pi_j : d_i^j \mapsto a_i^j \in A.$$

The agent also is either given or learns a set of higher-level options to reduce the time required to solve the task. Options are usually defined in the same state-space as the problem, so an option o would be defined as follows:

$$\begin{array}{rll} \pi_o : & (d_i^j, a) & \mapsto [0,1] \\ I_o : & d_i^j & \mapsto \{0,1\} \\ \beta_o : & d_i^j & \mapsto [0,1]. \end{array}$$

Options defined in this way are not portable between tasks because the form and meaning of d (as a problem-space descriptor) may change from one task to another. However, the form and meaning of c (as an agent-space descriptor) does not. Therefore we define agent-space option components as:

$$\begin{array}{rll} \pi_o : & (c_i^j, a) & \mapsto [0,1] \\ I_o : & c_i^j & \mapsto \{0,1\} \\ \beta_o : & c_i^j & \mapsto [0,1]. \end{array}$$

Although the agent is learning task and option policies in different spaces, because it receives both an agent-space sensation and a problem-space descriptor at each state both policies can be updated simultaneously.

4 Experiments

4.1 The Lightworld Domain

In the Lightworld domain, a robot is placed in an environment consisting of a sequence of rooms, with each room containing a locked door, a lock, and possibly a key. In order to leave a room, the robot must unlock the door and step through it. In order to unlock the door, it must move up to the lock and press it, but if a key is present in the room, the robot must be holding it to successfully unlock the door. To obtain the key, the robot must move on top of it and pick it up. The robot receives a reward of 1000 for leaving the door of the final room, and a step penalty of -1 for each action. The robot has six actions: movement in each of the four grid directions, a pickup action and a press action.

In addition, we equip the robot with twelve light sensors, grouped into threes on each of its sides. The first sensor in each triplet detects red light, the second green and the third blue. Each light sensor responds to light sources on its side of the robot, ranging from a reading of 1 when the robot is on top of the light source, to a reading of 0 when the light source is 20 squares away. Open doors emit a red light, keys on the floor (but not those held by the robot) emit a green light, and locks emit a blue light. An example lightworld is given in Figure 1.

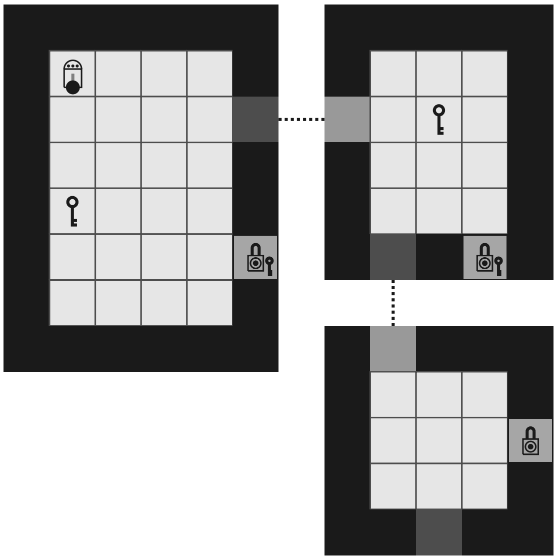

Figure 1: A small example lightworld.

For every particular lightworld instance, a problem-space descriptor requires five pieces of data: the current room number, the x and y coordinates of the robot in that room, whether or not the robot has the key, and whether or not the door is open. We use the light sensor readings as an agent-space because their semantics remain consistent across lightworld instances. In this case the agent-space (with 12 continuous variables) has higher dimension than any of the individual problem-spaces.

4.2 Types of Agent

We use five types of agents in our experiments. The standard reinforcement learning agent (without options) used Sarsa(λ) ($\alpha = 0.1$, $\gamma = 0.99$, $\lambda = 0.9$, $\epsilon = 0.01$) with each state assigned an initial value of 500.

The standard reinforcement learning agent (with options) used the same learning strategy but added an (initially unlearned) option for each pre-specified salient event (picking up each key, unlocking each lock, and walking through each door). Each option was in the same state-space and used the same learning parameters as the basic reinforcement learning agent, but used off-policy trace-based tree-backup updates (Precup et al., 2000) for intra-option learning. Each option got a reward of 1 when it completed successfully and used a discount factor of 0.99 per action, and could be taken only in the room in which it is defined, and in states where its value function exceeds a minimum threshold (0.0001). Because these options are learned in problem-space, they are useful but must be relearned each time because they cannot be ported between lightworld instances.

The standard reinforcement learning agent (with perfect options) was given already learned options for each salient event. It still performed option updates and was otherwise identical to the standard agent with options, but it represents an agent using ideal option transfer.

The reinforcement learning agents with agent-space options employed three options: one for picking up a key, one for going through an open door and one for unlocking a door. These options were defined in agent-space, so each option's value function was a function of the twelve light sensors, rather than a problem-space descriptor. Since these variables are continuous we used linear function approximation for each option's value function, performing updates using gradient descent ($\alpha = 0.01$) and off-policy trace-based tree-backup updates. The agent gave each option a reward of 1 upon completion, and used a step penalty of 0.05 and a discount factor of 0.99. An option could be taken at a particular state when its value function there exceeded a minimum threshold (0.1). Because these options are learned in agent-space, they can be ported between lightworld instances.

Finally, we tested reinforcement learning agents employing both agent-space and problem-space options simultaneously. Since these agents (like the other agents using options) learned using intra-option learning methods, they represent agents that learn both portable and general, and non-portable but specific and exact skills simultaneously.

4.3 Experimental Structure

To evaluate the performance of the agent types, we generated 100 random lightworlds. Each lightworld consisted of between 2 and 5 rooms, each having a width and height of between 5 and 15 cells. The door and lock in each room were randomly distributed around the room boundaries, and $\frac{1}{3}$ of the rooms included a randomly placed key. This results in state space sizes of between 600 and approximately 20,000 state-action pairs, with an average size of 4900 state-action pairs, slightly larger than the state space size of the Taxi domain (Dietterich, 2000), a standard hierarchical reinforcement learning test domain. We evaluated each problem-space option agent type on ten samples of each world, resulting in a total of 1000 samples.

To evaluate the changing performance of the agent-space options as the agents gained more experience, we again obtained ten samples for each world, resulting in 1000 total samples. For each test world, we ran the agents once without any training and then with a varying number of training experiences. A training experience in a given test world consisted of 100 episodes in an environment randomly selected from the remaining 99 lightworlds. Although the agents were allowed to update their options during evaluation in the test world, these updates were discarded after testing, so that the agent-space options were never given prior training in the same world they were being tested in.

4.4 Results

Figure 2 shows average learning curves for agents employing problem-space options, and Figure 3 shows the same for agents employing agent-space options. They show that the first time an agent-space option agent encounters a lightworld it performs in approximately the same way as an agent without options (as evidenced by two topmost learning curves in each figure), but its performance rapidly improves with experience in other lightworlds. After experiencing a single training lightworld the agent has a much shallower learning curve than an agent using problem-space options alone, until by 5 experiences its learning curve is similar to that of an agent with perfect problem-space options (compare with the bottom-most learning curve of Figure 2), even though its options are never trained in the same world in which it is tested. Thus the comparison between Figures 2 and 3 clearly shows that agent-space options can be successfully transferred between lightworld instances.

Figure 4 shows average learning curves for agents employing *both* types of options[1]. The first time such agents encounter a lightworld they perform as well as agents using problem-space options (compare with the second highest curve in Figure 2), and thereafter rapidly improve their performance, doing better than agents using only agent-space options, and again by 5 experiences performing nearly as well as agents with perfect options. We conjecture that this improvement results from two factors. First, the agent-space in our example is a much larger space than any individual problem-space, so problem-space options are much easier to learn from scratch than agent-space options. This explains why agents using only agent-space options and no training experiences perform more like agents without any options than like agents with problem-space options. Second, options learned in our problem-space can represent exact solutions to specific subgoals, whereas options learned in our agent-space must be approximated, and are likely to be general and therefore slightly less efficient for any specific subgoal. This explains why an agent using both types of options performs better in the long run than an agent using only agent-space options.

[1] We note that in eight of the more than $200,000$ episodes we used when testing agents with both types of options the function approximator used to represent the agent-space value functions diverged, and we restarted the episode. Although this is a known problem with the off-policy trace-based backup method we used (Precup et al., 2000), it did not occur at all during the same number of samples obtained for agents using agent-space options only.

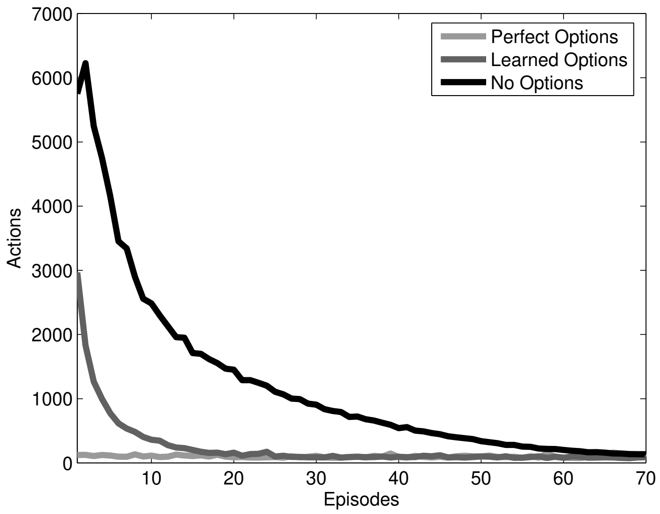

Figure 2: Learning curves averaged over 1000 lightworld instances for agents employing problem-space options.

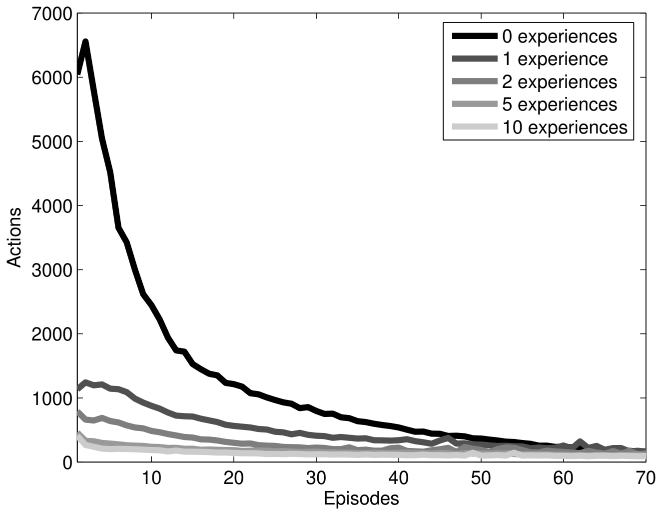

Figure 3: Learning curves averaged over 1000 lightworld instances for agents employing agent-space options, and having experienced varying numbers of training worlds.

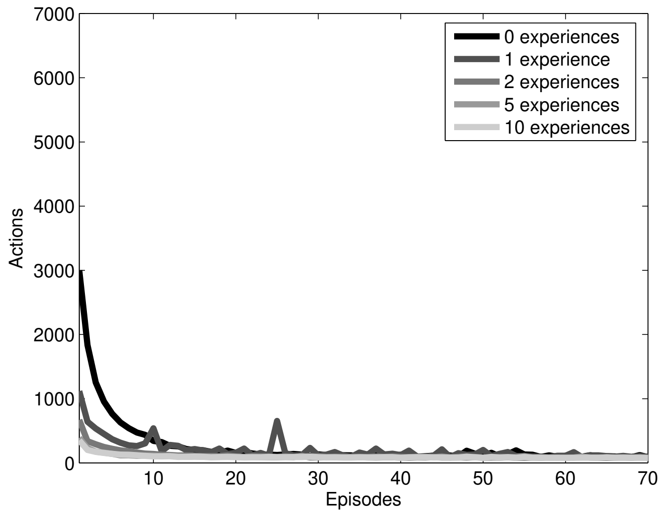

Figure 4: Learning curves averaged over 1000 lightworld instances for agents employing agent-space *and* problem-space options, having experienced varying numbers of training worlds.

Figure 5 shows the mean total number of steps required over 70 episodes for agents using no options, problem-space options, perfect options, agent-space options, and both option types. Again it shows that experience in training environments rapidly drops the number of total steps required, to nearly as low as the total number required for an agent given perfect options. It also clearly shows that agents using both types of options do consistently better than those using agent-space options alone. We note that the error bars in Figure 5 are small and decrease with increasing experience, indicating consistent transfer.

5 Discussion

In this paper we have introduced a framework for learning macro-actions in agent-space, a different space from the one in which the agent is solving a problem. The concept of an agent-centric representation is closely related to the notion of deictic or ego-centric representations (Agre & Chapman, 1987), where objects are represented from the point of view of the agent rather than in some global frame of reference. We expect that for most problems, especially in robotics, agent-space representations will be egocentric, except in manipulation tasks, where they will likely be object-centric. In problems involving spatial maps, we expect that the difference between problem-space and agent-space will be closely related to the difference between allocentric and egocentric representations of space (Guazzelli et al., 1998).

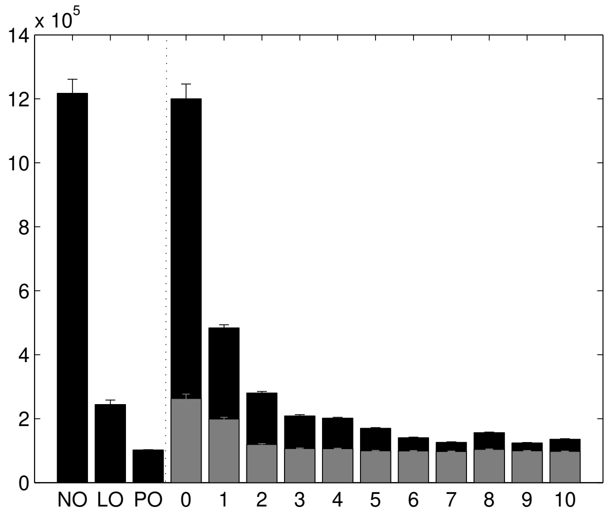

Figure 5: Total number of steps required for 70 episodes, averaged across 1000 lightworld instances, for reinforcement learning agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options having experienced varying numbers of training worlds (0-10, dark bars), and both option types having experienced varying numbers of training worlds (0-10, light bars).

We also expect that it will often be the case that the learning problem for an option in agent-space will actually be harder than solving an individual problem-space instance, as was the case in our experiments. In such situations, learning both types of options simultaneously is likely to improve performance. Since intra-option learning methods allow for several options to learn from the same experiences, it may be better in general to simultaneously learn both general portable skills and specific, exact but non-portable skills, and allow them to bootstrap each other.

The introduction of agent-space descriptors into the reinforcement learning framework creates a difficult design problem. Although this design problem is similar to that of standard state space design, researchers in the reinforcement learning community have so far developed significant expertise at designing problem-spaces, but not agent-spaces. We expect that the two design problems are equivalently difficult, and that with practice determining the relevant agent-spaces for related problems will become easier.

Finally, we note that although we have presented the notion of learning skills in agent-space using the options framework, the same idea could easily be used in other reinforcement learning frameworks, for example the MAXQ (Dietterich, 2000) or Hierarchy of Abstract Machines (HAM) (Parr & Russell, 1997) formulations.

6 Conclusion

In this paper we have introduced the notion of learning options in agent-space rather than problem-space as a mechanism for building portable high-level skills in reinforcement learning agents. Our experimental results show that such options can be successfully transferred between tasks that share an agent-space, and significantly improve performance in later tasks, both by themselves and in conjunction with more specific but non-portable problem-space options.

Acknowledgements

We would like to thank Sarah Osentoski, Özgür Şimşek, Aron Culotta, Ashvin Shah, Chris Vigorito, Kim Ferguson, Andrew Stout, Khashayar Rohanimanesh, Pippin Wolfe and Gene Novark for their comments and assistance. Andrew Barto and George Konidaris were supported by the National Science Foundation under Grant No. CCF-0432143, and Andrew Barto was supported by a subcontract from Rutgers University, Computer Science Department, under award number HR0011-04-1-0050 from DARPA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Agre, P., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI 87)* (pp. 268–272).

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Systems*, *13*, 41–77. Special Issue on Reinforcement Learning.

Barto, A., Singh, S., & Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the International Conference on Developmental Learning (ICDL 04)*.

Bernstein, D. (1999). *Reusing old policies to accelerate learning on new MDPs* (Technical Report UM-CS-1999-026). Department of Computer Science, University of Massachusetts at Amherst.

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

Digney, B. (1998). Learning hierarchical control structures for multiple tasks and changing environments. *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. Zurich, Switzerland: MIT Press.

Guazzelli, A., Corbacho, F., Bota, M., & Arbib, M. (1998). Affordances, motivations, and the world graph theory. *Adaptive Behavior*, *6*, 433–471.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 02)* (pp. 243–250).

Jonsson, A., & Barto, A. (2001). Automated state abstraction for options using the U-Tree algorithm. *Advances in Neural Information Processing Systems 13 (NIPS 01)* (pp. 1054–1060).

Jonsson, A., & Barto, A. (2005). A causal approach to hierarchical decomposition of factored MDPs. *Proceedings of the Twenty Second International Conference on Machine Learning (ICML 05)*.

Konidaris, G., & Barto, A. (2005). *Autonomous shaping: Learning to predict reward for novel states* (Technical Report UM-CS-2005-258). Department of Computer Science, University of Massachusetts at Amherst.

Mannor, S., Menache, I., Hoze, A., & Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. *Proceedings of the Twenty First International Conference on Machine Learning (ICML 04)* (pp. 560–567).

McGovern, A., & Barto, A. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 01)* (pp. 361–368).

Parr, R., & Russell, S. (1997). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10 (NIPS 97)* (pp. 1043–1049).

Perkins, T., & Precup, D. (1999). *Using options for knowledge transfer in reinforcement learning* (Technical Report UM-CS-1999-034). Department of Computer Science, University of Massachusetts, Amherst.

Pickett, M., & Barto, A. (2002). Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. *Proceedings of the Nineteenth International Conference of Machine Learning (ICML 02)* (pp. 506–513).

Precup, D., Sutton, R., & Singh, S. (2000). Eligibility traces for off-policy policy evaluation. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 00)* (pp. 759–766).

Ravindran, B., & Barto, A. (2002). Model minimization in hierarchical reinforcement learning. *Proceedings of the Fifth Symposium on Abstraction, Reformulation and Approximation (SARA 02)* (pp. 196–211).

Ravindran, B., & Barto, A. (2003). Relativized options: Choosing the right transformation. *Proceedings of the Twentieth International Conference on Machine Learning (ICML 03)* (pp. 608–615).

Şimşek, Ö., & Barto, A. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 04)* (pp. 751–758).

Şimşek, Ö., Wolfe, A., & Barto, A. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 05)*.

Singh, S., Barto, A., & Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS 04)*.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 98)* (pp. 556–564).

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.

Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning. *Advances in Neural Information Processing Systems* (pp. 385–392). The MIT Press.