Building Portable Options: Skill Transfer in Reinforcement Learning

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

Abstract

The options framework provides methods 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 solving, they cannot be used in other tasks that are similar but have different state spaces. We introduce the notion of learning options in agent-space, the space generated by a feature set 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 have different problem-spaces. We present experimental results demonstrating the use of agent-space options in building transferrable skills, and show that they perform best when used in conjunction with problem-space options.

1 Introduction

Much recent research in reinforcement learning has focused on hierarchical methods [Barto & Mahadevan, 2003] and in particular the *options* framework [Sutton et al., 1999], which integrates macro-action learning into the reinforcement learning framework. Most option learning methods work within the same state space as the problem the agent is solving at the time. Although this can lead to faster learning on later tasks in the same state space, learned options would be more useful if they could be reused in later tasks that are related but have distinct state spaces.

We propose learning portable options by using two separate representations: 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 dimensionality) [Konidaris & Barto, 2006]. Options learned in agent-space can be reused in future problem-spaces because the semantics of agent-space remain consistent across tasks.

We present the results of an experiment showing that learned agent-space options can significantly improve performance across a sequence of related but distinct tasks, but are best used in conjunction with problem-specific options.

2 Background

2.1 The Options Framework

An option o consists of three components:

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

where π_o is the *option policy* (a probability distribution over actions at each state in which the option 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 β_o is the *termination condition*, giving the probability of the option terminating in each state [Sutton et al., 1999]. The options framework provides methods for learning and planning using options as temporally extended actions in the standard reinforcement learning framework [Sutton & Barto, 1998].

Algorithms for learning new options must include a method 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 performed by an off-policy reinforcement learning algorithm so that the agent can update many option simultaneously after taking an action [Sutton et al., 1998].

Creation and termination are usually 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. Previous research has selected goal states by a varierty of methods (e.g., variable change frequency [Hengst, 2002], local graph partitioning [Şimşek et al., 2005] and salience [Singh et al., 2004]). Other research has focused on extracting options by exploiting commonalities in collections of policies over a single state space [Thrun & Schwartz, 1995; Bernstein, 1999; Perkins & Precup, 1999; Pickett & Barto, 2002].

All of these methods 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. The available state abstraction methods [Jonsson & Barto, 2001; Hengst, 2002] only allow for the automatic selection of a subset of this space for option learning, or require an explicit transformation from one space to another [Ravindran & Barto, 2003].

2.2 Sequences of Tasks and Agent-Space

We are concerned with an agent that is required to solve a sequence of related but distinct tasks, defined as follows. The agent experiences a sequence of environments generated by the same underlying 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 sufficient to distinguish Markov states in the current 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 each of these state spaces a *problem-space*.

The agent also uses a second representation based on the features that are consistently present and retain the same semantics across tasks. This space is shared by all of the tasks in the sequence, and we call it an *agent-space*. The two spaces stem from two different representational requirements: problem-space models the Markov description of a particular task, and agent-space models the (potentially non-Markov) commonalities across a sequence of tasks. We thus consider a sequence of tasks to be related if they 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 (2006) 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 is a sequence of buildings in each of which a robot equipped with a laser range finder is required to reach a target room. 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, thus forming a building-specific problem space. The 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 agent-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 in any building that the robot encounters later.

3 Options in Agent-Space

We consider an agent solving n problems, with state spaces $S_1, ..., S_n$, and 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), c_i^j is an agent-space descriptor, and r_i^j is the reward obtained at the state. The goal of reinforcement learning in each task S_j is to find a policy π_j that maximizes reward.

The agent is also either given, or learns, a set of higher-level options to reduce the time required to solve the task. Options defined using d 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:

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

Although the agent will be learning task and option policies in different spaces, both types of policies can be updated simultaneously as the agent receives both agent-space and problem-space descriptors at each state.

4 Experiments

4.1 The Lightworld Domain

An agent 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 agent 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 agent must be holding it to successfully unlock the door. The agent can obtain a key by moving on top of it and picking it up. The agent receives a reward of 1000 for leaving the door of the final room, and a step penalty of -1 for each action. Six actions are available: movement in each of the four grid directions, a pickup action and a press action. The environments are deterministic and unsuccessful actions (e.g., moving into a wall) result in no change in state.

We equip the agent 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 sensor responds to light sources on its side of the agent, ranging from a reading of 1 when it is on top of the light source, to 0 when it is 20 squares away. Open doors emit a red light, keys on the floor (but not those held by the agent) emit a green light, and locks emit a blue light. Figure 1 shows an example.

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

Types of Agent

We used five types of reinforcement learning agents: agents without options, agents with problem-space options, agents with perfect problem-space options, agents with agent-space options, and agents with both option types.

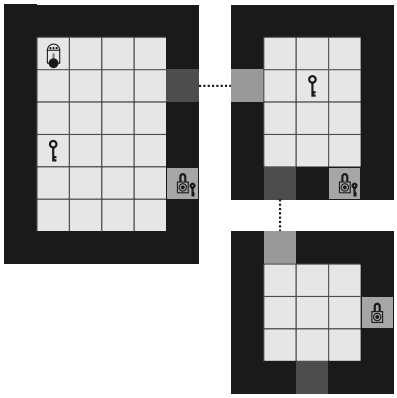

Figure 1: A small example lightworld.

The agents without options used Sarsa(λ) with ϵ-greedy action selection ($\alpha = 0.1$, $\gamma = 0.99$, $\lambda = 0.9$, $\epsilon = 0.01$) to learn a solution policy in problem-space, with each state-action pair assigned an initial value of 500.

Agents with problem-space options had an (initially unlearned) option for each pre-specified salient event (picking up each key, unlocking each lock, and walking through each door). Options were learned in problem-space and used the same parameters as the agent without options, but used off-policy trace-based tree-backup updates [Precup et al., 2000] for intra-option learning. Options got a reward of 1 when completed successfully, used a discount factor of 0.99 per action, and could be taken only in the room in which they were defined, and in states where their value function exceeds a minimum threshold (0.0001). Because these options are learned in problem-space, they are useful but must be relearned for each individual lightworld.

Agents with perfect problem-space options were given prelearned options for each salient event. They still performed option updates and were otherwise identical to the standard agent with options, but they represented agents with perfectly transferrable fully learned options.

Agents with agent-space options still learned their solution policies in problem-space but learned their option policies in agent-space. Each agent employed three options: one for picking up a key, one for going through an open door and one for unlocking a door, with each one's policy a function of the twelve light sensors. Since the sensor outputs 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 transferred between lightworld instances.

Finally, agents with both types of options were included to represent agents that learn both general portable and specific non-portable skills simultaneously.

Note that all agents used discrete problem-space value functions to solve the underlying task instance, because their agent-space descriptors are only Markov in one room lightworlds, which were not present in our experiments.

Experimental Structure

We generated 100 random lightworlds, each consisting of 2-5 rooms with width and height of 5-15 cells. A door and lock were were randomly placed on each room boundary, and $\frac{1}{3}$ of rooms included a randomly placed key. This resulted in state space with between 600 and approximately $20,000$ state-action pairs (4900 on average). We evaluated each problem-space option agent type on 1000 lightworlds (10 samples of each generated lightworld).

To evaluate the performance of agent-space options as the agents gained more experience we similarly obtained 1000 samples, but for each sample we ran the agents once without training and then with between 1-10 training experiences. Each training experience for a sample lightworld consisted of 100 episodes in a training lightworld randomly selected from the remaining 99. Although the agents updated their options during evaluation in the sample lightworld, these updates were discarded before the next training experience so the agent-space options never received prior training in the evaluation lightworld.

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. The first time an agent-space option agent encounters a lightworld it performs similarly to 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 lightworld in which it is tested. The comparison between Figures 2 and 3 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 (performing 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 is much larger than any individual problem-space, so problem-space options are easier to learn from scratch

[1] In 8 of the more than $200,000$ episodes used when testing agents with both types of options an agent-space value function approximator diverged and we restarted the episode. Although this is a known problem with the backup method we used [Precup et al., 2000], it did not occur at all during the same number of samples obtained for agents with agent-space options only.

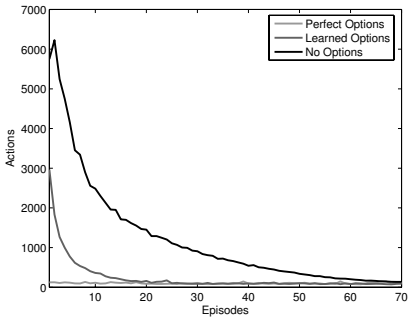

Figure 2: Learning curves for agents with problem-space options.

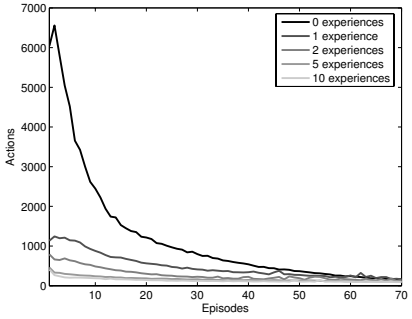

Figure 3: Learning curves for agents with agent-space options, with varying numbers of training experiences.

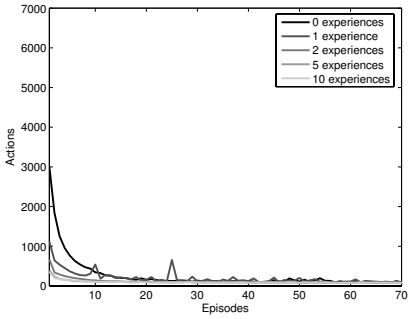

Figure 4: Learning curves for agents with agent-space *and* problem-space options, with varying numbers of training experiences.

than agent-space options. This explains why agents using only agent-space options and no training experiences perform more like agents without 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 are general and must be approximated, and are therefore likely to be slightly less efficient for any specific subgoal. This explains why agents using both types of options perform better in the long run than agents using only agent-space options.

Figure 5 shows the mean total number of steps required

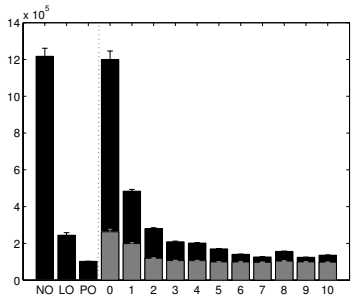

Figure 5: Total steps over 70 episodes for agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options with 0-10 training experiences (dark bars), and both option types with 0-10 training experiences (light bars).

over 70 episodes for agents using no options, problem-space options, perfect options, agent-space options, and both option types. Experience in training environments rapidly drops the number of total steps required to nearly as low as the number required for an agent with 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 experience, indicating consistent transfer.

4.2 The Conveyor Belt Domain

A conveyor belt system must move a set of objects from a row of feeders to a row of bins. There are two types of objects (triangles and squares) and each bin starts has a capacity for each type. The objects are issued one at a time from a feeder and must be directed to a bin. Dropping an object into a bin with a positive capacity for its type decrements that capacity.

Each feeder is directly connected to its opposing bin through a conveyor belt, which is connected to the belts above and below it at a pair of fixed points along its length. The system may either run the conveyor belt (which moves the current object one step along the belt) or try to move it up or down (which only moves the object if it is at a connection point). Each action results in a penalty of -1, except where it causes an object to be dropped into a bin with spare capacity, in which case it results in a reward of 100. A small example conveyor belt system is shown in Figure 6.

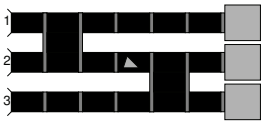

Figure 6: A small example conveyor belt problem.

Each system has a camera that tracks the current object and returns values indicating the distance (up to 15 units) to the bin and each connector along the current belt. Because the space generated by the camera is present in every conveyor-

belt problem and retains the same semantics it is an agent-space, and because it is discrete and relatively small ($13,500$ states) we can learn policies in it without function approximation. However, because it is non-Markov (due to its limited range and inability to distinguish between belts) it cannot be used as a problem-space.

A problem-space descriptor for a conveyor belt instance consists of three numbers: the current object number, the belt it is on and how far along that belt it lies (technically we should include the current capacity of each bin, but we can omit this and still obtain good policies). We generated 100 random instances with 30 objects and 20-30 belts (each of length 30-50) with randomly selected interconnections, resulting in problem-spaces of $18,000$-$45,000$ states.

We ran experiments where the agents learned three options: one to move the current object to the bin at the end of the belt it is currently on, one for moving it to the belt above it, and one for moving it to the belt below it. We used the same agent types and experimental structure as before, except that the agent-space options did not use function approximation.

Results

Figures 7, 8 and 9 show learning curves for agents employing no options, problem-space options and perfect options; agents employing agent-space options; agents employing both types of options, respectively.

Figure 8 shows that the agents with agent-space options and no prior experience initially improve quickly but eventually obtain lower quality solutions than agents with problem-space options (Figure 7). One or two training experiences result in roughly the same curve as agents using problem-space options but by 5 training experiences the agent-space options are a significant improvement, although due to their limited range they are never as good as perfect options. This initial dip is probably due to the limited range of the agent-space options (due to the limited range of the camera) and the fact that they are only locally Markov, even for their own subgoals.

Figure 9 shows that agents with both option types do not experience this initial dip, and outperform problem-space options immediately, most likely because the agent-space options are able to generalise across belts. Figure 10 shows the mean total reward for each type of agent. Agents using agent-space options eventually outperform agents using problem-space options only, even though the agent-space options have a much more limited range; agents using both types of options consistently outperform agents with either option type and eventually approach the performance of agents using pre-learned problem-space options.

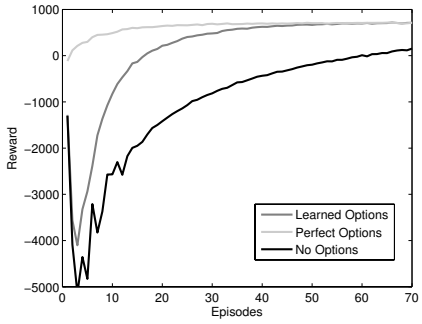

Figure 7: Learning curves for agents with problem-space options.

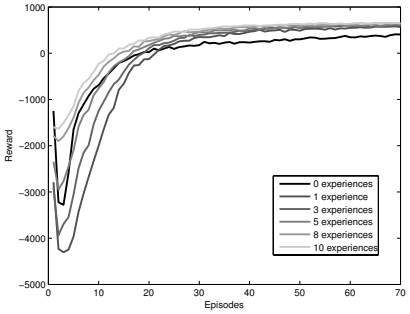

Figure 8: Learning curves for agents with agent-space options, with varying numbers of training experiences.

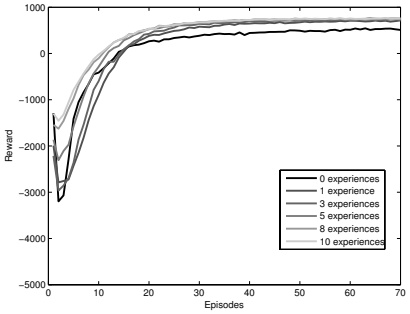

Figure 9: Learning curves for agents with both types of options, with varying numbers of training experiences.

5 Discussion

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].

We also expect that learning an option in agent-space will often 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 the update of several options 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 addition of agent-space descriptors to the reinforcement learning framework introduces a design problem similar

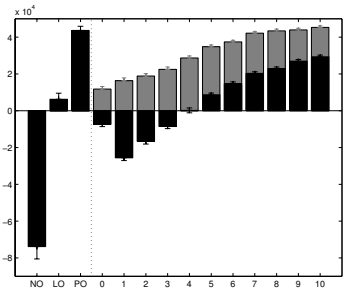

Figure 10: Total reward over 70 episodes for agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options with 0-10 training experiences (dark bars), and both option types with 0-10 training experiences (light bars).

to that of standard state space design. Additionally, for option learning an agent-space descriptor should ideally be Markov within the set of states that option is defined over. The agent-space descriptor form will therefore affect both what options can be learned and their range. In this respect designing agent-spaces for learning options requires more care than for learning shaping functions [Konidaris & Barto, 2006].

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

6 Conclusion

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

Acknowledgements

We 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 in part by the National Science Foundation under Grant No. CCF-0432143, and Andrew Barto was supported in part by a subcontract from Rutgers University, Computer Science Department, under award number HR0011-04-1-0050 from DARPA.

References

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

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.

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* (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* (pp. 1054–1060).

Konidaris, G., & Barto, A. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. *Proceedings of the Twenty Third International Conference on Machine Learning* (pp. 489–496).

Parr, R., & Russell, S. (1997). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10* (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* (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* (pp. 759–766).

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

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*.

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

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* (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.