Local Graph Partitioning as a Basis for Generating Temporally-Extended Actions in Reinforcement Learning

Özgür Şimşek
University of Massachusetts
Amherst, MA 01003-9264
ozgur@cs.umass.edu

Alicia P. Wolfe
University of Massachusetts
Amherst, MA 01003-9264
pippin@cs.umass.edu

Andrew G. Barto
University of Massachusetts
Amherst, MA 01003-9264
barto@cs.umass.edu

Abstract

We present a new method for automatically creating useful temporally-extended actions in reinforcement learning. Our method identifies states that lie between two densely-connected regions of the state space and generates temporally-extended actions (e.g., options) that take the agent efficiently to these states. We search for these states using graph partitioning methods on local views of the transition graph. This local perspective is a key property of our algorithms that differentiates it from most of the earlier work in this area, and one that allows it to scale to problems with large state spaces.

Introduction

Reinforcement learning (RL) researchers have recently developed several formalisms that address planning, acting, and learning at multiple levels of temporal abstraction. These include Hierarchies of Abstract Machines (Parr & Russell 1998; Parr 1998), MAXQ value function decomposition (Dietterich 2000), and the options framework (Sutton, Precup, & Singh 1999; Precup 2000). (See Barto & Mahadevan, 2003, for a recent review.) These formalisms pave the way toward dramatically improved capabilities of autonomous agents, but to fully realize their benefits, an agent needs to be able to create useful temporal abstractions automatically instead of relying on a system designer to provide them.

A number of methods have been suggested for addressing this need. One approach is to search for commonly occurring subpolicies in solutions to a set of tasks and to define temporally-extended actions with corresponding policies (Thrun & Schwartz 1995; Pickett & Barto 2002). Another approach is to identify subgoal states—states that are useful to reach—and generate temporally-extended actions that take the agent efficiently to these states. Subgoals considered useful include states that are visited frequently or that have a high reward gradient (Digney 1998), states that are visited frequently on successful trajectories but not on unsuccessful ones (McGovern & Barto 2001), and states that lie between densely-connected regions of the state space (Menache, Mannor, & Shimkin 2002; Şimşek & Barto 2004; Mannor *et al.* 2004).

In this paper, we propose a new method for temporal abstraction in RL based on identifying subgoal states. Our subgoals are those states that lie between two densely-connected regions of the state space, for example a doorway between two rooms. Şimşek & Barto (2004) call such states *access states*; we adopt their terminology here.

The utility of access states has been argued previously in the literature (McGovern & Barto 2001; Menache, Mannor, & Shimkin 2002; Şimşek & Barto 2004; Mannor *et al.* 2004). Their main appeal is allowing more efficient exploration of the state space by leading the agent to regions that it would not go to frequently when performing a random walk, which characterizes the initial stage of many RL algorithms. Furthermore, because these subgoals are defined independently of the reward function, they are useful in solving not only the current task but also a variety of other tasks that share the same state transition matrix but differ in their reward structure—getting to the doorway is a useful thing to do regardless of what the agent needs to do in the other room.

The main distinction between this work and methods proposed earlier in the literature is in how the search for access states is conducted. We perform this search by periodically constructing and examining a local view of the MDP transition graph, i.e., one that reflects only the most recent experiences of the agent. We accept as subgoals those states that are consistently part of the cut identified when a graph partitioning algorithm is applied to this view of the transition graph.

Our method is similar to QCut (Menache, Mannor, & Shimkin 2002) in that both use graph-theoretic methods to find cuts of the transition graph and use them to identify subgoals of interest. QCut, however, takes a global view of the transition graph, using the entirety of the agent's experience to construct the graph. This distinction between the two algorithms, while subtle, gives rise to two fundamentally different algorithms.

We use a local view of the transition graph because the access states—states that lie between two densely-connected regions of the state space—are defined in reference to what surrounds them rather than their global position in the transition graph. An access state may or may not be part of a global cut of the whole graph. For example, leaving one's house is an access state, but within the context of one's entire state space, it probably will not be part of a global cut.

The local perspective we have is shared with RN (Şimşek & Barto 2004), an algorithm that never constructs the transition graph, but works with only the most recent part of the transition history in identifying the same type of subgoals. We provide a more detailed discussion of similarities and differences among QCut, RN, and our method in the discussion section.

We call our algorithm LCut, emphasizing its local view of the transition graph. In the following sections we describe LCut in detail, evaluate its performance in two domains, and conclude with a discussion of our results, related work, and future directions.

Description of the Algorithm

Our algorithm consists of periodically performing the following steps, which will be explained in detail in the following sections.

1. Construct a graph that corresponds to the agent's most recent state transitions.
2. Apply a graph partitioning algorithm to identify a cut that partitions this graph into two densely-connected blocks that have relatively few edges between them.
3. If any state that is part of the identified cut meets the subgoal evaluation criteria, identify it as a subgoal.
4. For each new subgoal state, create a temporally-extended action that takes the agent efficiently to this state.

Building a Partial Transition Graph

LCut periodically constructs a partial transition graph using the recent transition history. This graph is weighted and directed, and is constructed in a straightforward manner given a transition sequence: Vertices in the graph correspond to the states in the transition sequence; edges correspond to transitions between these states; edge weights equal to the number of corresponding transitions that take place in the transition sequence. As noted earlier, the use of only a recent part of the transition history is an essential part of our algorithm. The length of the transition sequence to be used in building this graph is a parameter of the algorithm (*h*).

Finding a Cut

After constructing a partial transition graph, LCut seeks to find a cut through this graph that partitions it into two densely-connected blocks that have relatively few edges between them. Below we describe the cut evaluation criteria and the spectral clustering technique used to find a good cut.

Cut Evaluation Criterion The ideal cut would partition the states into two blocks in which the transition probability within blocks is high and the transition probability across blocks is low. There is a well known cut evaluation metric that provides this property: Normalized Cut (NCut) (Shi & Malik 2000).

The original NCut measure was intended for undirected graphs; we modify it to include directed edges. For a graph partitioned into blocks *X* and *Y*, let e_{ij} be the weight on the edge from vertex *i* to vertex *j*, *cut*(*X*, *Y*) be the sum of the weights on edges that originate in *X* and end in *Y*, and *vol*(*X*) be the sum of weights of all edges that originate in *X*. We define NCut as follows:

$$\text{NCut} = \frac{cut(A,B)}{vol(A)} + \frac{cut(B,A)}{vol(B)} \qquad (1)$$

Our choice of NCut is motivated by a relatively recent finding by Meila & Shi (2001) that relates NCut to the probability of crossing the cut during a random walk on an undirected graph. More precisely, for an undirected graph, each term in Equation 1 corresponds to the probability of crossing the cut in one step in a random walk that starts from the corresponding block, if the start state within the block is selected with respect to the stationary distribution of the graph's Markov Chain.

These results do not generally apply to directed graphs, but the special structure of the graph we are working with—it represents frequency of state transitions in a Markov chain—allows us to derive a similar property. For our graph, this first term in Equation 1 equals the following:

$$\frac{cut(A,B)}{vol(A)} = \frac{\sum_{i\in A, j\in B} t_{ij}}{\sum_{i\in A} t_{ij}}. \qquad (2)$$

where t_{ij} is the total number of sampled transitions from state *i* to state *j*. In other words, this first term is an estimate of the probability that the agent transitions to block *B* in one step given that it starts in block *A*.

A similar argument can be made for the second term in Equation 1. The NCut value gives equal weight to both blocks, regardless of their size. As a consequence, NCut gives us the sum of probabilities of crossing the cut from each block. This property of NCut makes it particularly well suited for our problem—partitioning the graph such that transitioning between blocks has a low probability and transitioning within blocks has a high probability.

We note here that there are two alternative cut metrics that are commonly used in graph partitioning: MinCut (Ahuja, Magnati, & Orlin 1993) and RatioCut (Hagen & Kahng 1992). MinCut is the sum of edge weights that form the cut, while RatioCut is defined as follows for an undirected graph:

$$RatioCut = cut(A,B)/|A| + cut(B,A)/|B|. \qquad (3)$$

Neither of these two metrics provides as compelling a reason as NCut to be used as our evaluation metric. MinCut, in particular, creates a bias towards cuts that separate a small number of nodes from the rest of the graph—for example a single corner state in a gridworld—and is clearly inferior to the other two metrics.

Finding a Partition That Minimizes NCut Finding a partition of a graph that minimizes NCut is NP-hard (Shi & Malik 2000). LCut finds an approximate solution using a spectral clustering algorithm, as described in Shi & Malik (2000), which has a running time of $O(N^3)$, where *N* is the number of vertices in the graph.

Subgoal Evaluation Criteria

At this point, we would like to remind the reader that LCut operates on an approximation of the transition graph which reflects only the most recent transitions of the agent. This implies that the same region of the state space will generate a different transition graph each time the agent visits it. As a consequence, the first rule that comes to mind in evaluating subgoals—if the cut quality is good, accept as subgoals all vertices that participate in the cut—will not be effective. It will accept those states that truly lie between densely-connected regions, but also those that *appear* to do so in the latest sample.

We will need to deal with noise and the tool at our disposal is repeated sampling. Let *targets* be those states that actually lie between densely-connected regions, and *hits* be those states that are part of a cut returned by the partitioning algorithm. Because targets will be more likely to be hits than non-targets, over repeated samples, a target will be a hit relatively more often than non-targets.

In fact, assuming independent, identically-distributed sampling of a partial transition graph, the number of hits follows a Binomial distribution, with a success probability that is higher for targets than for non-targets, and what we face is a classification task that aims to distinguish targets from non-targets. This is a simple classification task (Duda, Hart, & Stork 2001) that has the following optimal decision rule:

Label state as target if

$$\frac{n_t}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}} \cdot \frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}} \tag{4}$$

where n_t is the number of times the state was part of a cut returned by the partitioning algorithm, n is the number of times this state was part of the input graph to the partitioning algorithm, p is the probability that a target will be part of a cut, q is the probability that a non-target will be part of a cut, λ_{fa} is the cost of a false alarm, λ_{miss} is the cost of a miss, $p(N)$ is the prior probability of a target, and $p(T)$ is the prior probability of a non-target.

This decision rule is a simple threshold on the proportion of visitations in which the state was part of a cut returned by the partitioning algorithm. The first term on the right is a constant that depends only on class-conditional probabilities. The second term depends in addition on the number of observations, the priors, and the relative cost of each type of error. Since this term is inversely related to the number of times the state is visited, its influence decreases with increasing number of visits to the state.

While we can not use Rule 4 directly—we do not know the values of many of the quantities in this equation—we use it to motivate the following algorithm: Accept a state as subgoal only if it has been part of the transition graph some threshold number of times (t_v) and if the proportion of times it was part the resulting cut in these graphs is greater than some threshold value (t_p).

Another consequence of the way we construct the transition graph is missing edges. This may occasionally result in one of the terms in Equation 1 to be zero, giving the corresponding block a perfect score, only because none of the edges that go from this block to the other one are observed. This will lead to substantially lower than actual estimates of the cut quality and is not desirable. To avoid this, we use the Laplace correction in computing each term in Equation 1, adding one to the number of edges within the block and to the number of edges going out to the other block.

Generating Temporally-Extended Actions

In defining temporally-extended actions, we adapt the options framework (Precup 2000; Sutton, Precup, & Singh 1999). A (Markov) *option* is a temporally-extended action, specified by a triple $\langle I, \pi, \beta \rangle$, where I denotes the option's initiation set, i.e., the set of states in which the option can be invoked, π denotes the policy followed when the option is executing, and $\beta : I \rightarrow [0, 1]$ denotes the option's termination condition, with $\beta(s)$ giving the probability that the option terminates in state $s \in I$.

When a new subgoal is identified, LCut generates an option whose policy efficiently takes the agent to this subgoal from any state in the option's initiation set. The option's initiation set is determined using those transition sequences that returned the subgoal state as part of a cut; it consists of those states that were visited shortly before the subgoal in these sequences and that ended up in the same partition as the subgoal. How many past transitions to include in this set is determined by a parameter, the *option lag* (l_o).

The option's policy is specified through an RL process employing action replay (Lin 1992) and a reward function specific to the subgoal for which the option was created (corresponding to what Dietterich, 2000, called a *pseudo reward function*). The reward function that LCut uses causes a policy to be learned that makes the agent reach the subgoal state in as few time steps as possible while remaining in the option's initiation set. This is achieved by giving a large positive reward for reaching the subgoal, a large negative reward for exiting the initiation set, and a small negative reward for every transition.

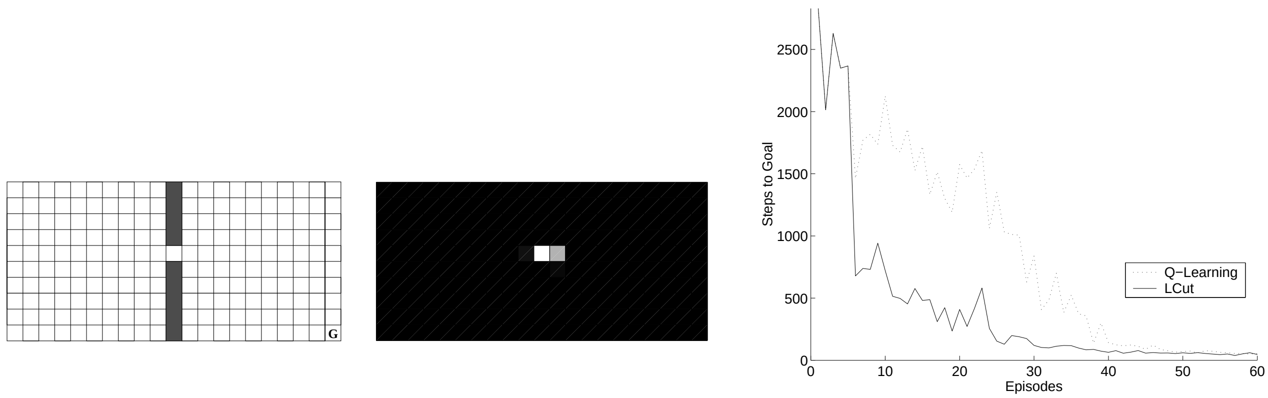
And finally, the option's termination condition specifies that the option terminates with probability 1 if the agent reaches the subgoal, or if the agent leaves the initiation set; otherwise, it terminates with probability 0.

Algorithmic Complexity of the Subgoal Identification Method

The time complexity of LCut is O(h^3), where h is the length of the transition sequence used to construct the transition graphs. The running time does not grow with the size of the state space because the algorithm always works with a bounded set of states, regardless of the size of the actual state space. This is a key property of the algorithm that allows it to scale to larger domains.

Experimental Results

The first question we would like to answer is whether the algorithm is able to identify states that lie between densely-connected regions as intended. To answer this question,



(a) (b) (c)

Figure 1: (a) Two-Room gridworld environment, (b) Subgoals identified, (c) Mean steps to goal.

we present performance results in two domains, a two-room gridworld and the taxi task introduced by Dietterich (2000).

In our experiments, the agent used Q-learning with ϵ-greedy exploration with $\epsilon = 0.1$. The learning rate (α) was kept constant at 0.05; initial Q-values were 0. The parameters of the algorithm was set as follows: $t_c = 0.05$, $t_p = 0.5$, $t_v = 5$ for the two-room gridworld task, $t_v = 10$ for the taxi task. These settings were based on our intuition and simple trial and error; we are currently investigating various methods of setting these parameters automatically. No limit was set on the number of options that could be generated; and no filter was employed to exclude certain states from being identified as subgoals.

Two-Room Gridworld

This domain is shown in figure 1a. The agent started each episode on a random square in the west room; the goal was the grid square on the southeast corner of the grid. The four primitive actions, `north`, `south`, `east`, and `west`, moved the agent in the intended direction with probability 0.9, and in a uniform random direction with probability 0.1. If the direction of movement was blocked, the agent remained in the same location. The agent received a reward of 1 at the goal state, and a reward of 0 at all other states. The discount factor was 0.9.

Figure 1b shows a visual representation of the location and frequency of the subgoals identified by the algorithm in 30 runs. The color of a square in this figure corresponds to the number of times it was identified as a subgoal, with lighter colors indicating larger frequencies. The state that was identified as a subgoal most frequently was the doorway, in 26 of the 30 runs. A total of 47 subgoals were identified (1.6 subgoals per run on average), 46 of which were within one step of the doorway.

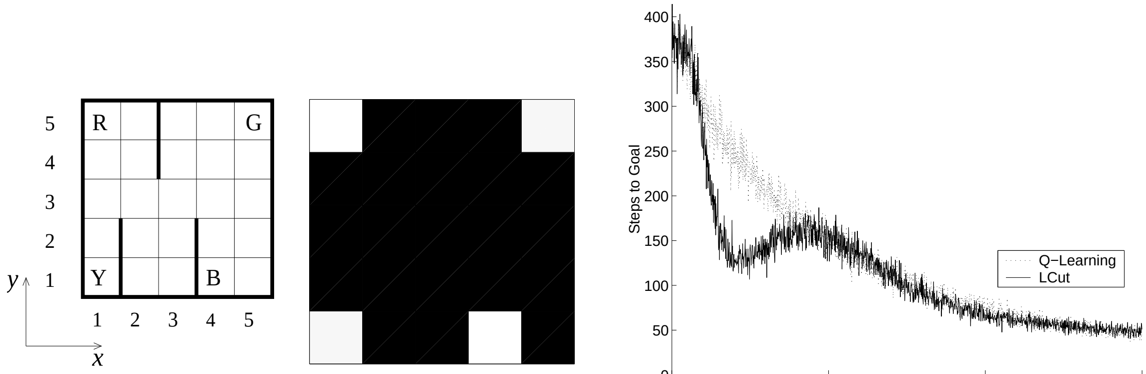
Figure 1c shows the mean number of steps taken to reach the goal state. LCut was able to identify useful subgoals and show a marked improvement in performance compared to plain Q-learning within 5 episodes.

Taxi Task

In the taxi domain the task is to pick-up and deliver a passenger to her destination on a 5×5 grid depicted in Figure 2a. There are four possible source and destination locations: the grid squares marked with R, G, B, Y. The source and destination are randomly and independently chosen in each episode. The initial location of the taxi is one of the 25 grid squares, picked uniformly random. At each grid location, the taxi has a total of six primitive actions: `north`, `east`, `south`, `west`, `pick-up`, and `put-down`. The navigation actions succeed in moving the taxi in the intended direction with probability 0.80; with probability 0.20, the action takes the taxi to the right or left of the intended direction. If the direction of movement is blocked, the taxi remains in the same location. The action `pick-up` places the passenger in the taxi if the taxi is at the same grid location as the passenger; otherwise it has no effect. Similarly, `put-down` delivers the passenger if the passenger is inside the taxi and the taxi is at the destination; otherwise it has no effect. Reward is -1 for each action, an additional +20 for passenger delivery, and an additional -10 for an unsuccessful `pick-up` or `put-down` action. Successful delivery of the passenger to the destination marks the end of an episode.

This domain has 500 states: 25 grid locations, 5 passenger locations (including in-taxi), and 4 destinations. There are two types of states in this domain that conform to our definition of a subgoal. The first is a consequence of the sequential nature of the task. In order to succeed, the taxi needs to go to the passenger location, pick up the passenger, go to the destination, and drop off the customer, in sequence. The completion of any of these subtasks is a subgoal we would like to identify. The second type of subgoal is navigational. Even though the grid is very small, the walls in the domain limit navigation quite a bit, and grid squares (2,3) and (3,3) act as navigational bottlenecks.

The performance of the algorithm was evaluated over 100 runs. Figure 2b shows a visual representation of the grid location of the subgoals, ignoring the other two state variables. The mean number of subgoals identified per run was 10.9. All of these corresponded to driving to the passenger



(a) (b) (c)

Figure 2: (a) The taxi domain, (b) Subgoals identified (showing only the grid location variables), (c) Mean steps to goal.

location.

Mean number of steps to complete the task is shown in Figure 2c. The figure reveals an early improvement in performance in comparison to Q-learning, but this improvement does not continue. We are unable to explain why this is the case.

Discussion

In both experimental tasks, LCut successfully identified target states as subgoals. These initial results suggest that the algorithm succeeds at finding cuts of the actual transition graph even though it works with incomplete samples. These results are promising, but further study of the algorithm is needed to conclude its general effectiveness. One important research direction is to have the agent adaptively set the algorithm parameters that we set here heuristically.

LCut is closely related to a number of algorithms proposed in the literature, most notably to QCut (Menache, Mannor, & Shimkin 2002) and RN (Şimşek & Barto 2004). All three algorithms search for the same type of subgoals—states that lie between two densely-connected regions of the state space—but differ in how they search for such states. QCut constructs the MDP transition graph and applies a min-cut/max-flow algorithm to identify a minimum cut of the graph. The main distinction between QCut and LCut is the scope of the transition graph they construct. QCut constructs the entire transition graph of the underlying MDP, reflecting the entirety of the agent's experience, and finds cuts through this global graph. In contrast, LCut constructs a local view of the graph and perform cuts on this small region. This difference between the algorithms has two implications. First, they are expected to identify different states as subgoals because a local cut may or may not be a global cut of the entire transition graph. We expect that LCut will be more successful at identifying access states—access states are part of local cuts but not necessarily of global ones. And second, the running time of LCut's subgoal discovery method does not grow with the size of the state space, while QCut's subgoal discovery method has time complexity $O(N^3)$, where N is the number of nodes in the graph.

As a side, we would like to note here that the spectral clustering algorithm used here may also be incorporated into QCut. QCut perform cuts using a min-cut/max-flow algorithm, but evaluates the quality of the cuts using a different metric, RatioCut, because of the drawbacks of the MinCut metric discussed earlier. Incorporating spectral clustering algorithms into QCut would allow the cuts to be created using the actual evaluation metric (RatioCut) and would stop the need for specifying a source and a sink for the min cut/max flow algorithm.

RN and LCut are similar in that they both conduct their search using only the most recent part of the transition history. RN never constructs a transition graph, but uses a heuristic that uses a measure of relative novelty to identify subgoal states. An advantage RN has over LCut is its algorithmic simplicity—the running time of its subgoal discovery method has a time complexity of O(1). We may think of RN as using a simple heuristic to approximate what LCut is doing. Assessing the relative strengths and weaknesses of these two algorithms is an important research direction.

Acknowledgments

This work was supported by the National Science Foundation under Grant No.ECS-0218123 to Andrew G. Barto and Sridhar Mahadevan. 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. Alicia P. Wolfe was partially supported by a National Physical Science Consortium Fellowship and funding from Sandia National Laboratories.

References

Ahuja, R. K.; Magnati, T. L.; and Orlin, J. B. 1993. *Network Flows Theory: Algorithms and Applications*. Prentice Hall Press.

Barto, A. G., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4):341 – 379.

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

Digney, B. 1998. Learning hierarchical control structure for multiple tasks and changing environments. In *From Animals to Animats 5: The Fifth Conference on the Simulation of Adaptive Behaviour*. MIT Press.

Duda, R. O.; Hart, P. E.; and Stork, D. G. 2001. *Pattern Classification*. New York: Wiley.

Hagen, L., and Kahng, A. B. 1992. New spectral methods for ratio cut partitioning and clustering. In *IEEE Trans. Computer-Aided Design*, volume 11, 1074–1085.

Lin, L. 1992. Self-Improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8:293–321.

Mannor, S.; Menache, I.; Hoze, A.; and Klein, U. 2004. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*. Forthcoming.

McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In Brodley, C. E., and Danyluk, A. P., eds., *Proceedings of the Eighteenth International Conference on Machine Learning*, 361–368. Morgan Kaufmann.

Meila, M., and Shi, J. 2001. Learning segmentation with random walk. In *Neural Information Processing Systems (NIPS) 2001*.

Menache, I.; Mannor, S.; and Shimkin, N. 2002. Q-Cut - Dynamic discovery of sub-goals in reinforcement learning. In Elomaa, T.; Mannila, H.; and Toivonen, H., eds., *Proceedings of the Thirteenth European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, 295–306. Springer.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In Jordan, M. I.; Kearns, M. J.; and Solla, S. A., eds., *Advances in Neural Information Processing Systems*, volume 10, 1043–1049. MIT Press.

Parr, R. 1998. *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. Dissertation, Computer Science Division, University of California, Berkeley.

Pickett, M., and Barto, A. G. 2002. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In Sammut, C., and Hoffmann, A. G., eds., *Proceedings of the Nineteenth International Conference on Machine Learning*, 506–513. Morgan Kaufmann.

Precup, D. 2000. *Temporal abstraction in reinforcement learning*. Ph.D. Dissertation, University of Massachusetts Amherst.

Shi, J., and Malik, J. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

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

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

Thrun, S., and Schwartz, A. 1995. Finding structure in reinforcement learning. In Tesauro, G.; Touretzky, D. S.; and Leen, T. K., eds., *Advances in Neural Information Processing Systems*, volume 7, 385–392. MIT Press.