2005

Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning

Özgür Şimşek
University of Massachusetts - Amherst

Alicia P. Wolfe
University of Massachusetts - Amherst

Andrew G. Barto
University of Massachusetts - Amherst

Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning

Özgür Şimşek OZGUR@CS.UMASS.EDU
Alicia P. Wolfe PIPPIN@CS.UMASS.EDU
Andrew G. Barto BARTO@CS.UMASS.EDU
Department of Computer Science, University of Massachusetts, Amherst, MA 01003-9264 USA

Abstract

We present a new subgoal-based method for automatically creating useful skills in reinforcement learning. Our method identifies subgoals by partitioning *local* state transition graphs—those that are constructed using only the most recent experiences of the agent. The local scope of our subgoal discovery method allows it to successfully identify the type of subgoals we seek—states that lie between two densely-connected regions of the state space—while producing an algorithm with low computational cost.

1. Introduction

Recent methods in Reinforcement Learning (RL) allow an agent to plan, act, and learn with temporally-extended actions (Dietterich, 2000; Parr, 1998; Precup, 2000; Sutton et al., 1999). A temporally-extended action, or a *skill*, is a closed-loop policy over one-step actions, for example one that takes a robot to its battery charger using lower level sensory and motor actions. A suitable set of skills can help improve an agent's efficiency in learning to solve difficult problems. If an agent can develop such skill sets automatically, it should be able to efficiently solve a variety of problems without relying on hand-coded skills tailored to specific problems.

A number of methods have been suggested towards this end. One approach is to search for commonly occurring subpolicies in solutions to a set of tasks and to generate skills with corresponding policies (Pickett & Barto, 2002; Thrun & Schwartz, 1995). A second approach is to identify subgoals—states that are useful to reach—and to learn skills that take the agent efficiently to these subgoals. Subgoals proposed in the literature 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 (Mannor et al., 2004; Menache et al., 2002; Şimşek & Barto, 2004). In addition, Hengst (2002) has used the notion of a subgoal in performing temporal and spatial abstraction simultaneously, defining subgoals to be those states that lead to transitions that the agent can not correctly represent or predict at the current level of state abstraction.

We propose a new subgoal-based method for learning skills in RL. We define our subgoals in terms of two regions of the state space that have the following property: transitioning from one region to the other in one step has a low, but strictly positive, probability, and most of these transitions go through a small set of states. The states in this set are our subgoals. A simple example is a doorway between two rooms: all transitions from one room to the other go through the doorway.

Our subgoal definition is very similar to those of Mannor et al. (2004), Menache et al. (2002), and Şimşek and Barto (2004); we adopt the terminology of Şimşek and Barto (2004) and call them *access states*. The main appeal of access states is that they allow more efficient exploration of the state space by providing easy access to neighboring regions. Furthermore, because access states 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 functions—getting to the doorway is useful regardless of what the agent needs to do in the other room.

Appearing in *Proceedings of the 22st International Conference on Machine Learning*, Bonn, Germany, 2005.

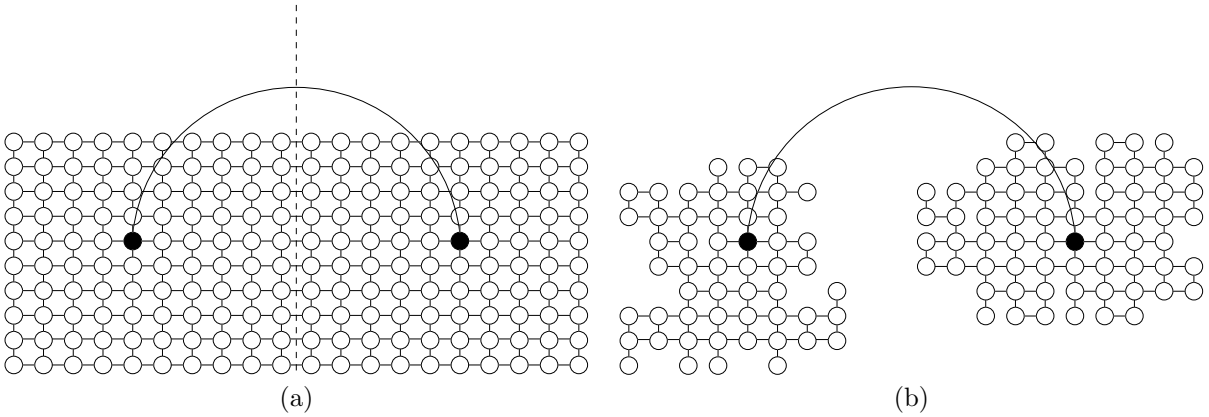


Figure 1. (a) The state transition graph of a simple gridworld domain. The dashed line shows a cut of the graph that minimizes *NCut*.(b) A sample local transition graph in this domain.

The main distinction between our method and those that were proposed earlier is in how access states are identified. Our method identifies access states by periodically constructing a *local transition graph* (a state-transition graph that reflects only the most recent experiences of the agent), finding a cut of this graph with a low between-blocks transition probability, and accepting as subgoals those states that are endpoints of edges that consistently cross identified cuts. We call our method L-Cut, emphasizing its *local* perspective and its graph-theoretic basis.

Key to our subgoal discovery method is the local scope of the transition graph. The cuts that are identified are not cuts of the entire state space but of only a small part of it encountered recently. This local perspective parallels how access states are defined—locally, in relation to the states that surround them, rather than in relation to the entire state space—and is essential in correctly identifying them. Methods that use cuts of the entire state space., e.g., Q-Cut (Menache et al. 2002), will not be able to correctly identify these states because an access state may or may not be part of a global cut.

The only other existing subgoal discovery method that shares the local perspective of L-Cut is the Relative Novelty Algorithm (RN) of Şimşek and Barto (2004). L-Cut and RN both use the most recent part of the transition history in identifying access states, but differ in how they do this. L-Cut takes a graph-theoretic approach, while RN uses a heuristic measure of novelty. In the discussion section, we provide a more detailed comparison of L-Cut, RN, and other subgoal-based approaches to automatically creating temporally-extended actions.

In the following sections we expand on the utility of local cuts, describe L-Cut in detail, empirically evaluate its performance, and conclude with a discussion of our results, related work, and future directions.

2. The Utility of Local Cuts

We illustrate the utility of local cuts using a simple gridworld with the state transition graph shown in Figure 1(a). All edges are bi-directional; edge directions and weights are omitted from the figure. In addition to north, south, east, and west transitions, this domain includes a shortcut between the two states colored in black. These two states are access states—they provide the only one-step transition from their vicinity to the other part of the grid.

The dashed line in Figure 1(a) shows a cut of this graph using edge weights of one and the *NCut* metric (which we discuss in the following sections). This cut is not useful in identifying the access states: While the two access states border an edge that crosses the cut, so do an additional 20 states that do not conform to our subgoal definition. In fact, any cut of this graph will either not include the edge between the access states or will have a large number of additional states bordering the cut edges.

We show in Figure 1(b) what a local transition graph in this domain might look like. The figure communicates clearly the utility of a local perspective: A cut of this graph is likely to single out the edge between the two access states. Of course, not all transition sequences will yield a graph that looks like this figure. Some will not include the shortcut edge at all, many will show a different edge connecting two otherwise unconnected sets of states of about equal size, and others will have no clear separation between two sets of states. How to deal with noise due to sampling

is an important problem that needs to be addressed. L-Cut does this by combining the evidence from an ensemble of local transition graphs as we describe in the following sections.

The argument for local cuts extends beyond this simple example that we have created to convey the intuition behind our method. Most real-world problems will show a complex connectivity structure that will not lend itself to the use of global cuts in identifying the access states. For example, analogous situations exist in continuous control problems where sequential composition of "funnels" in system dynamics can give rise to access-like states (Burridge et al., 1999).

We note here an alternative use of local cuts to identify access states: Build the entire transition graph, but perform cuts on local neighborhoods, for example on a part of the graph that contains only the states that are within a certain distance of a randomly selected state. This approach would be effective in identifying the access states and, unlike L-Cut, would not have to address any issues that result from sampling the graph. It would, however, require the agent to maintain the entire state transition graph and therefore would not scale well to problems with large state sets.

3. Description of the Algorithm

L-Cut is an iterative algorithm. Each iteration takes as input a state trajectory—a sequence of states visited by the agent while an RL algorithm is executing or in exploration mode, for example as the agent performs a random walk. Using this trajectory, the agent constructs a corresponding local transition graph, finds a cut of this graph such that the transition probability within blocks is high but between blocks is low, and, if any state qualifies as a subgoal, generates a skill that takes the agent efficiently to this state. Iterations are performed periodically, after the agent experiences a certain number of transitions and also at the end of each episode if the task is episodic. The input state trajectory is the one experienced since the last iteration of the algorithm.

3.1. Constructing the Local Transition Graph

The local transition graph is a weighted and directed graph constructed from the input state trajectory. Vertices in the graph correspond to the states in the trajectory; edges correspond to transitions between these states. Edge weights are equal to the number of corresponding transitions that take place in the trajectory.

3.2. Cut Metric

Given a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, a cut (A, B) of G is a partition of V; the edges that cross the cut are those with one endpoint in block A and the other in block B. We seek to minimize the *Normalized Cut* metric (*NCut*) (Shi & Malik, 2000), defined as follows:

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

where $cut(A, B)$ is the sum of the weights on edges that originate in A and end in B, and $vol(A)$ is the sum of weights on all edges that originate in A.

Our choice of *NCut* as a cut evaluation metric is not arbitrary. For a local transition graph, the first term in Equation 1 is the number of observed transitions from a state in block A to a state in block B, divided by the total number of transitions from states in block A. This is an estimate of the probability that the agent transitions to block B in one step given that it starts in block A under its current policy. A similar argument can be made for the second term in Equation 1. Their sum, *NCut*, is an estimate of the sum of probabilities of crossing the cut from each block, a metric 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.

Alternative cut metrics commonly used in graph partitioning are *MinCut* (Wu & Leahy, 1993) and *RatioCut* (Hagen & Kahng, 1992). *MinCut* is the sum of edge weights that cross the cut, while *RatioCut* equals $cut(A, B)/|A|+cut(A, B)/|B|$ for an undirected graph. Neither of these meets our needs as well as *NCut* does. *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.

3.3. Partitioning Algorithm

Finding a partition of a graph that minimizes *NCut* is NP-hard, but there exist good approximate algorithms for undirected graphs. We use the spectral clustering algorithm of Shi and Malik (2000), computing *NCut* on an undirected version of the local transition graph, in which edge weights show the number of transitions in either direction. In other words, we approximate our true metric shown in Equation 1 with the *NCut* value in the undirected graph, which takes the following form if stated in terms of the quantities in Equation 1:

$$\widehat{NCut}(A,B) = \frac{cut(A,B)+cut(B,A)}{vol(A)+cut(B,A)} + \frac{cut(B,A)+cut(A,B)}{vol(B)+cut(A,B)} \quad (2)$$

Let us first consider how the first term in Equation 1 is influenced by this approximation. Both the numerator and the denominator are incremented by the same amount which will cause the value of this first term to be greater in the approximation than in the true metric. (Recall that in the true metric this term represents a proportion and is therefore between 0 and 1.) In general, the difference can be arbitrarily large, but for small values of *NCut* we expect it to be small—if *NCut* is small, cut sizes will typically be much smaller than the volume of the blocks. The second term is influenced in a similar manner. Because we seek cuts with small *NCut*, Equation 2 is a reasonable approximation.

We next briefly describe the partitioning algorithm. Let N be the number of vertices in the graph, w_{ij} be the weight on the edge between vertices i and j, $\mathbf{D}$ be a diagonal matrix with $D(i,i) = \sum_j w_{ij}$, and $\mathbf{W}$ be a matrix with $W(i,j) = w_{ij}$. Shi and Malik (2000) show that, for an undirected graph, the second eigenvalue of the matrix $\frac{(\mathbf{D}-\mathbf{W})}{\mathbf{D}}$ gives an approximation of the minimum *NCut* value and that the second eigenvector gives an approximation of the corresponding partition labels. Ideally the eigenvectors would take two integer values that tell us exactly how to split the nodes, but typically they take continuous values which requires us to choose a splitting point. Following Shi and Malik (2000), we compute *NCut* for the $N-1$ possible partitions consistent with the eigenvector ordering and select the partition with the lowest *NCut* value.

This algorithm has a running time of $O(N^3)$. When using L-Cut, N will typically be much smaller than the total number of states in the MDP because L-Cut constructs a local transition graph that shows only a small part of the agent's state trajectory.

3.4. Subgoal Evaluation Criteria

A cut of the local transition graph with a low *NCut* value (below a threshold value t_c) indicates a good separation between the blocks, and those states that are endpoints of the edges that cross the cut are subgoal candidates. We call these states *hits*. Accepting all hits as subgoals is not an effective strategy. In addition to the access states of the domain, it identifies as subgoals a number of other states that look like access states in the sample transition graph. This is a consequence of using short trajectory samples to construct the graph. Even in a domain with no access states, for example a square gridworld, a local transition graph may have a cut with a low *NCut* value.

We need to be able to differentiate those states that are access states of the domain from those that *appear* to be so in the current local transition graph. In other words, we need to deal with noise, and the tool at our disposal is repeated sampling. Let *targets* be the access states in the domain. Because targets are more likely to be hits than non-targets, over repeated samples, a target is a hit relatively more often than a non-target. In fact, assuming independent, identically-distributed sampling, the number of times a given state qualifies as a hit follows a Binomial distribution, with a success probability that is higher for targets than for non-targets. If we make the simplifying assumption that all targets have the same success probability p, and all non-targets have the same success probability q, the optimal decision rule is to label a state as target if the following inequality holds (Duda et al., 2001) :

$$\frac{n_t}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}}\cdot\frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}}, \quad (3)$$

where n_t is the number of times the state was a hit, n is the number of observations on this state (i.e., the number of times the state was part of the sample transition graph), p is the probability that a target is a hit, q is the probability that a non-target is a hit, λ_{fa} is the cost of a false alarm, λ_{miss} is the cost of a miss, $p(T)$ is the prior probability of a target, and $p(N)$ is the prior probability of a non-target.

Inequality 3 is a threshold on the proportion of times a state was a hit when it was part of the sample graph. The first term on the right hand side is a constant; the second term is inversely related to the number of observations, therefore its influence decreases with increasing number of observations. While we can not use it directly—we do not know the values of many of the quantities involved—we use it to motivate the following algorithm: Accept a state as subgoal only if the number of observations on this state (n) is above a threshold value (t_o) and the proportion of observations in which the state was a hit is greater than some threshold value (t_p).

3.5. Generating Skills

We represent skills using the options framework (Precup, 2000; Sutton et al., 1999). When a new subgoal is identified, L-Cut generates an option whose policy

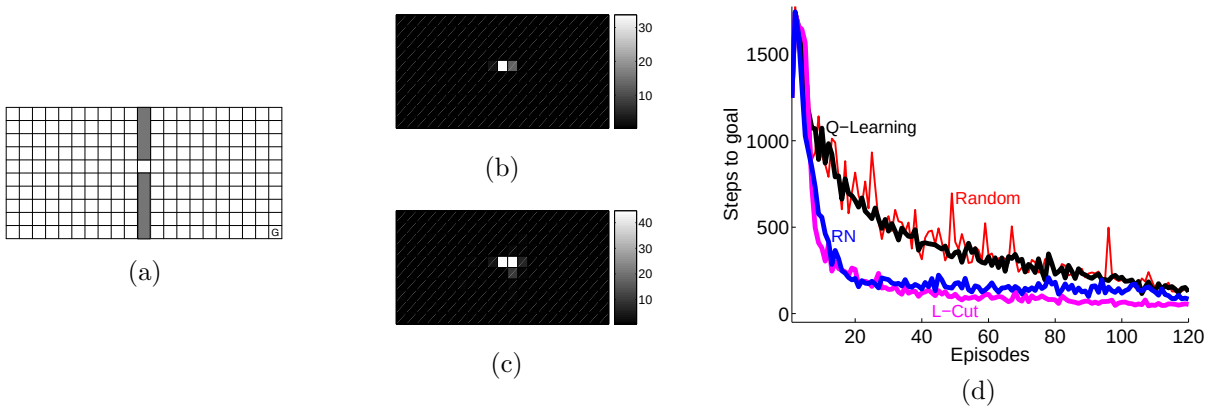


Figure 2. (a) Two-room gridworld environment. (b) Subgoals identified by L-Cut. (c) Subgoals identified by RN. (d) Mean steps to goal.

efficiently takes the agent to this subgoal. The option's initiation set is specified using those cuts that identified the subgoal as a hit. It includes those states that were in the opposite block and whose distance in the sample graph to the subgoal was less than *option lag* (l_o), a parameter of the algorithm. The option's policy is specified through an RL process employing action replay (Lin, 1992) using a pseudo reward function (Dietterich, 2000). The policy learned takes the agent to the subgoal state in as few transitions as possible while remaining in the option's initiation set. After the agent starts using the option, the algorithm continues to improve the option's policy using the new experiences of the agent (although it is not necessary to continue adjusting the option's policy after it becomes stable). 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.

3.6. Algorithmic Complexity

The time complexity of a single iteration of L-Cut is $O(h^3)$, where h is the length of the state trajectory used to construct the sample 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 makes it possible to scale to problems with large state sets.

4. Experimental Results

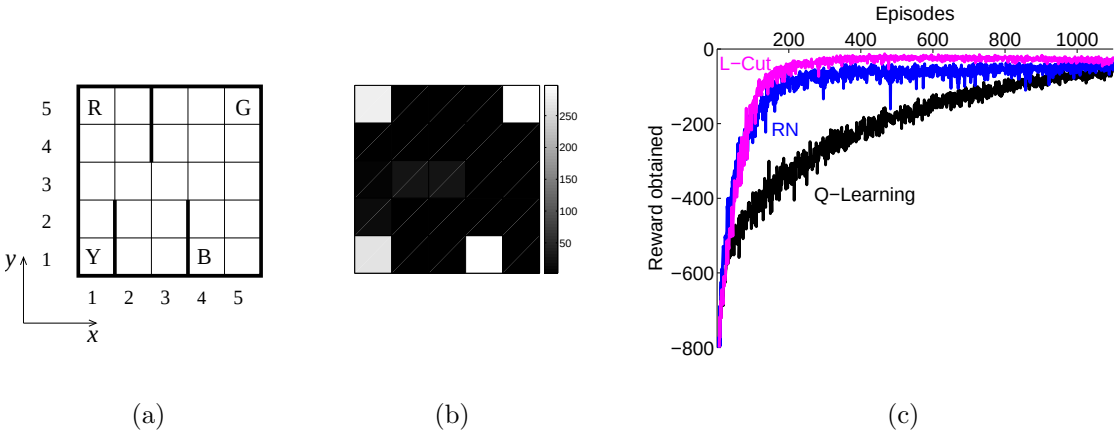
We present an empirical evaluation of L-Cut aimed at understanding whether L-Cut is effective in identifying the access states in an environment and whether the skills it generates are useful. We present results in two environments: a two-room gridworld and the taxi domain introduced by Dietterich (2000).

In our simulations, the agent used intra-option 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 agent started the trials with the availability of only primitive actions; no options were pre-defined. The parameters of L-Cut were set as follows: $h = 500$, $l_o = 10$, $t_c = 0.05$, $t_p = 0.25$, $t_o = 10$. These settings were based on our intuition; developing methods for setting them automatically is a topic of current research. We did not set any limits on the number of options that could be generated; nor did we employ any filters to exclude certain states from being identified as subgoals.

4.1. Two-Room Gridworld

We first present results in a two-room gridworld shown in Figure 2(a). This domain allows us to examine the performance of L-Cut in an environment we understand well. In addition, it allows us to compare it to an idealized version of RN—the only other local method for identifying access states—using the parameter settings reported in Şimşek and Barto (2004) which optimize its performance in this particular domain.

The agent started each episode on a random square in the west room. The goal was the grid square in 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 re-



(a) (b) (c)

Figure 3. (a) The taxi domain. (b) Subgoals identified by L-Cut. (c) Reward obtained.

mained in the same location. The agent received a reward of 1 at the goal state and a small negative reward (10^{-6}) at all other states. We found that the action penalty is important for L-Cut to perform well (also improving the performance of the baseline Q-learning algorithm). In its absence, the agent receives little feedback from the environment in early stages of learning and does not prefer one action over another. The action penalty allows the agent to take into account the fact that options have a higher cost of execution than primitive actions due to their longer execution time.

Figure 2(b) shows a visual representation of the location and frequency of the subgoals identified by L-Cut in 50 trials. The shading of a square in this figure corresponds to the number of times it was identified as a subgoal, with lighter colors indicating larger frequencies. L-Cut identified exactly one subgoal in each trial. In 34 of the trials, this was the doorway; in the remaining trials it was one of the two states that are one transition away from the doorway. For comparison, Figure 2(c) shows the subgoals identified by RN (mean = 2.40 subgoals per trial).

Figure 2(d) shows the mean number of steps taken to reach the goal state. The figure compares the performance of L-Cut to three other algorithms: RN, Q-learning with no option discovery process, and Random. Random refers to an agent that picked subgoals randomly: With each state transition, with a small probability p, the agent identified its current state as a subgoal and generated an option, employing a procedure similar to those used by RN and L-Cut. This random subgoal selection was started after 2000 transitions in the environment, so that the agent could gather enough experience to set the option policies correctly through action replay, and continued until 40,000 transitions were experienced. This period roughly corresponded to the period during which RN and L-Cut identified their subgoals. Having a fixed number of transitions for random subgoal selection allowed us to set p so that the expected number of options matched the mean number of options generated by L-Cut. The figure shows that random subgoal selection was not effective, while the subgoals identified by L-Cut and RN allowed the agent to learn the task much more efficiently. The figure also shows that L-Cut slightly improved on the performance of RN, particularly in later episodes.

4.2. Taxi Task

The task in this next domain is to pick-up and deliver a passenger to her destination on a 5×5 grid depicted in Figure 3(a). Source and destination are selected uniformly at random from among the grid squares marked with R, G, B, Y. The initial location of the taxi is one of the 25 grid squares, picked uniformly at random. At each grid location, the taxi has 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 marks the end of

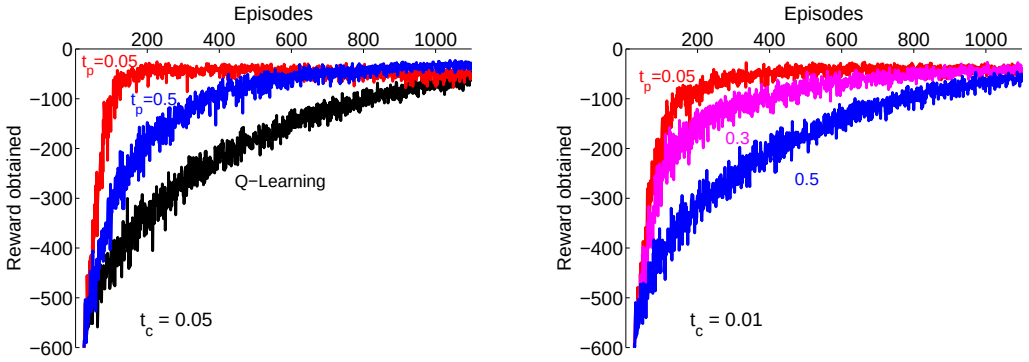


Figure 4. Sensitivity analysis in the taxi domain.

an episode. The domain has 500 states: 25 grid locations, 5 passenger locations (including in-taxi), and 4 destinations.

We evaluated the performance of L-Cut in 100 trials. Figure 3(b) is a visual representation of the grid locations of the subgoals identified, ignoring the other two state variables. The mean number of subgoals identified in a trial was 12.23. Of these, 93.2% corresponded to arriving at the passenger location or picking up the passenger; 4.3% were grid squares (2,3) and (3,3)—the main navigational bottlenecks in the domain. Figure 3(c) shows learning curves for L-Cut, Q-learning, and RN (using the same parameter settings as in Section 4.1). The results are similar to those obtained in the previous domain: Both L-Cut and RN showed an early improvement in performance in comparison to Q-learning, and L-Cut performed slightly better than RN in later episodes.

4.3. Sensitivity Analysis

It is important to examine the behaviour of L-Cut under various settings of its key parameters: t_c, the threshold on cut quality, and t_p, the threshold on the number of hits required for a state to qualify as a subgoal. The former parameter defines a threshold on the probability of transitioning between blocks in the sample graph. Higher values will cause more states to qualify as subgoals. This intuitive interpretation makes setting t_c relatively easy. Furthermore, we expect its ideal setting to be fairly consistent between domains.

In the taxi domain, we experimented with t_c=0.05, 0.01 and t_p=0.05, 0.1, 0.2, 0.3, 0.4, 0.5. For both settings of t_c, we observed a gradual decrement in performance with increasing values of t_p. We present some of the learning curves in Figure 4, including the best and the worst for both settings of t_c. All settings of the parameters either improved on or replicated the performance of the baseline Q-learning algorithm.

5. Discussion

L-Cut is closely related to a number of algorithms proposed in the literature, most notably to Q-Cut (Menache et al., 2002), RN (Şimşek & Barto, 2004), and the abstraction method of Mannor et al. (2004). All four algorithms search for the same type of subgoals but differ in how they do this search. The main distinction between Q-Cut and L-Cut is the scope of the transition graph they construct. Q-Cut constructs the entire transition graph of the underlying MDP, reflecting the entirety of the agent's experience, and finds cuts of this global graph. In contrast, L-Cut constructs a local view of the graph and performs cuts on this small part. This distinction between the two algorithms, while subtle, gives rise to two fundamentally different algorithms and 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. Second, the running time of L-Cut's subgoal discovery method does not grow with the size of the state space, while Q-Cut's subgoal discovery method has time complexity $O(N^3)$, where N is the number of states visited. The scope of the transition graph is also the key difference between L-Cut and the abstraction method of Mannor et al. (2004) which performs clustering on the entire transition graph.

We note that the spectral clustering algorithm we used may also be incorporated into Q-Cut. Q-Cut performs cuts using a min-cut/max-flow algorithm but evaluates the quality of the cuts using *RatioCut*. Incorporating a spectral clustering algorithm into Q-Cut would allow the cuts to be created using the actual evaluation

metric (*RatioCut*) and would eliminate the need for specifying a source and a sink for the min cut/max flow algorithm.

L-Cut is similar to RN in that both methods search for access states 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 L-Cut is its algorithmic simplicity—the running time of its subgoal discovery method has a time complexity of O(1).

The overall utility of our method will depend on the degree to which access states exist in the difficult real-world problems for which there is a need to scale reinforcement learning. But the results we have presented suggest that L-Cut is effective in identifying access states and that the temporally-extended actions it generates are useful in problems where such states exist.

Acknowledgments

This work was supported by the National Science Foundation under Grant No.CCF-0432143 and 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. This research was sustained in part by fellowship support from the National Physical Science Consortium Fellowship and Sandia National Laboratories.

References

Burridge, R., Rizzi, A. A., & Koditschek, D. E. (1999). Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, *18*, 534–555.

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. *From Animals to Animats 5: The Fifth Conference on the Simulation of Adaptive Behaviour*. MIT Press.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. New York: Wiley.

Hagen, L., & Kahng, A. B. (1992). New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *11*, 1074–1085.

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

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., & Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. *Proceedings of the Twenty-First International Conference on Machine Learning*. Banff, Alberta, Canada: ACM Press.

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

Menache, I., Mannor, S., & Shimkin, N. (2002). Q-Cut - Dynamic discovery of sub-goals in reinforcement learning. *Proceedings of the Thirteenth European Conference on Machine Learning* (pp. 295–306). Springer.

Parr, R. (1998). *Hierarchical control and learning for markov decision processes*. Doctoral dissertation, Computer Science Division, University of California, Berkeley.

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

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, University of Massachusetts Amherst.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*, 888–905.

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

Sutton, R. S., Precup, D., & Singh, S. P. (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). MIT Press.

Wu, Z., & Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*, 1101–1113.