Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning

Özgür Şimşek **Alicia P. Wolfe** **Andrew G. Barto**
Department of Computer Science
University of Massachusetts
Amherst, MA 01003-9264
{ozgur,pippin,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 that take the agent efficiently to these states. We search for these states using a graph partitioning algorithm on local estimates of the transition graph—those that are constructed using only the most recent experiences of the agent. This local perspective is a key property of our algorithm and one that differentiates it from most of the earlier work in this area.

1 Introduction

Reinforcement learning (RL) researchers have recently developed several formalisms that address planning, acting, and learning with temporally-extended actions. These include Hierarchies of Abstract Machines [1, 2], MAXQ value function decomposition [3], and the options framework [4, 5]. 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 temporally-extended actions automatically instead of relying on a system designer to provide them.

A number of methods have been suggested to address 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 [6, 7]. A second approach is to identify subgoals—states that are useful to reach—and generate temporally-extended actions 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 [8], states that are visited frequently on successful trajectories but not on unsuccessful ones [9], and states that lie between densely-connected regions of the state space [10, 11, 12].

We propose a new method for generating temporally-extended actions in RL based on identifying subgoal states. 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 similar to those of [10, 11, 12]; we adopt the terminology of Şimşek & Barto [11] and call them *access states*. The utility of access states as subgoals has been argued previously in the literature [9, 10, 11, 12]. Their main appeal is that they allow more

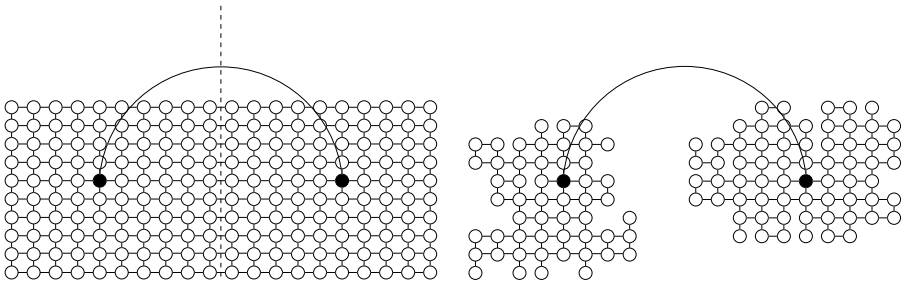


Figure 1: (a) The state transition graph of a simple gridworld domain. The dashed line shows a cut of the graph (see text). (b) A sample local view of this transition graph.

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.

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 the identified cut. The local scope of the transition graph is key to our subgoal discovery method. 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., QCut [10]) are not able to correctly identify these states because an access state may or may not be part of a global cut. Emphasizing the local perspective of our method, we call it LCut (which stands for Local Cut).

The only other subgoal discovery method that shares the local perspective of LCut is RN (the Relative Novelty Algorithm [11]). LCut and RN both use the most recent part of the transition history in identifying access states, but differ in how they do this. LCut takes a graph-theoretic approach, while RN uses a heuristic measure of novelty. In the discussion section, we provide a more detailed comparison of LCut, 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 LCut in detail, evaluate its performance in a simulated domain, 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 1a. All edges are bi-directional; we omit the directions and edge weights in 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 1a shows a cut of this graph using edge weights of one and the

NCut metric (which we discuss later). This cut is not useful in identifying the access states because a large number of additional states border the edges that cross the cut.

We show in Figure 1b 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; LCut does this by combining the evidence from a ensemble of local transition graphs as we describe in the following sections.

The argument for local cuts extends beyond this simple example. Most real-world navigation tasks have similar access states—elevators, highways, train stations, airports all impose shortcuts on top of a more regular grid structure. In general, 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. Analogous situations exist in continuous control problems where, for example, the sequential composition of "funnels" in system dynamics can give rise to access-like states [13].

3 Description of the Algorithm

LCut is an iterative algorithm that can be used while an RL algorithm is executing or in exploration mode, for example as the agent performs a random walk. Each iteration takes as input a state trajectory (i.e., a sequence of states), and 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 temporally-extended action that takes the agent efficiently to this state. Iterations are performed periodically, after a certain number of transitions (also at the end of each episode for episodic tasks). The input state trajectory is the one experienced since the last iteration of the algorithm. We describe each step in detail below.

Constructing the Local Transition Graph The local transition graph is a weighted, directed graph constructed from a 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.

Finding a Cut 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. LCut uses the Normalized Cut (NCut) [14] metric to evaluate the quality of a cut. Finding a partition of a graph that minimizes NCut is NP-hard [14]; LCut finds an approximate solution using a spectral clustering algorithm, as described in [14]. This algorithm has a running time of $O(N^3)$, where N is the number of vertices in the graph. When using LCut, N will typically be much smaller than the total number of states in the MDP, because LCut constructs, rather than the entire transition graph, a *local* transition graph that shows only a small part of the agent's state trajectory.

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

$$\text{NCut} \quad = \quad \frac{cutsize(A,B)}{vol(A)} + \frac{cutsize(B,A)}{vol(B)}. \tag{1}$$

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 a state 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. The NCut value, therefore, 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.

In practice, we found it useful to use the Laplace correction in computing each term in Equation 1, which adds one to the number of edges within the block and to the number of edges going out to the other block. The Laplace correction prevents either term from evaluating to zero, a perfect score, when the sample graph has no edges from the corresponding block to the other one.

We note here that there are two alternative cut metrics that are commonly used in graph partitioning: MinCut [15] and RatioCut [16]. MinCut is the sum of edge weights that cross the cut, while RatioCut equals $cutsize(A,B)/|A| + cutsize(A,B)/|B|$ 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.

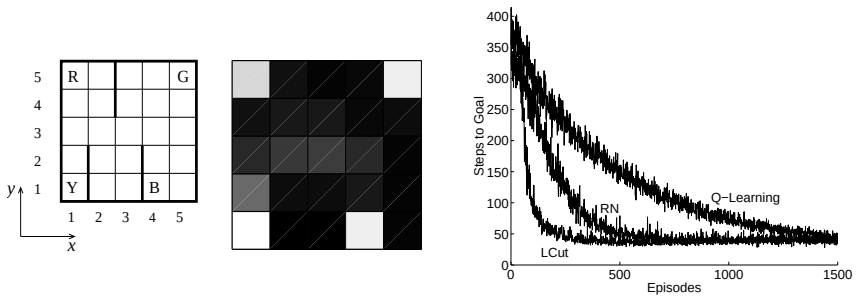
Subgoal Evaluation Criteria A cut of the local transition graph with a low NCut value (below a threshold value t_c) indicates a good separation between the blocks, and those states that are endpoints of the edges that cross the cut are subgoal candidates. We call these states *hits*. Accepting all hits as subgoals will not be effective: in addition to the access states of the domain, this will identify 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 with no walls, a local transition graph may return a cut with a low NCut value, making some states appear to be access states.

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 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 local 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 [17] that has the following optimal decision rule:

Label state as target if

$$\frac{n_t}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}} \cdot \frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}} \tag{2}$$

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 will be a hit, q is the probability that a non-target will be a hit, λ_{fa}



(a) (b) (c)

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

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.

This decision rule is a simple 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 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. This term is inversely related to the number of observations, therefore its influence decreases with increasing number of observations.

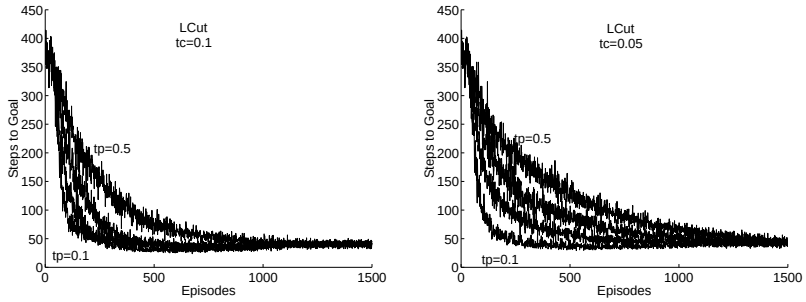
While we can not use Rule 2 directly—we do not know the values of many of the quantities in this equation—we use it to motivate the following algorithm: Accept a state as subgoal only if the number of observations on this state (n) is above a threshold value (t_o) and if the proportion of observations in which the state was a hit is greater than some threshold value (t_p).

Generating Temporally-Extended Actions In defining temporally-extended actions, we adopt the options framework [5, 4]. When a new subgoal is identified, LCut generates an option whose policy 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 the *option lag* (l_o), a parameter of the algorithm. The option's policy is specified through an RL process employing action replay [18] using a pseudo reward function [3]. The policy learned takes the agent to the subgoal state in as few time steps as possible while remaining in the option's initiation set. 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.

The time complexity of a single iteration of LCut 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.

4 Experimental Results

We present experimental results in a simulated taxi task introduced in [3]. 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 selected in each episode. The initial location



(a) (b)

Figure 3: Mean steps to goal in the taxi task using LCut (a) with t_c=0.05, t_p=0.1, 0.3, 0.4, 0.5, (b) with t_c=0.1, t_p=0.1, 0.3, 0.4, 0.5.

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. It includes two types of states that conform to our definition of access states: completion of subtasks (arriving at the passenger location and picking up the passenger) and navigational bottlenecks (grid squares (2,3) and (3,3))). Under various pairings of passenger location and destination, subtask subgoals amount to 32 states and navigational subgoals to 40 states.

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 LCut were set as follows: $h = 500$, $l_o = 10$, $t_c = 0.05$, $t_p = 0.1$, $t_o = 10$. These settings were based on our intuition; developing methods for setting them automatically is a topic of current research. No limit was set on the number of options that could be generated; no filter was employed to exclude certain states from being identified as subgoals.

The performance of the algorithm was evaluated over 100 runs. Figure 2b is a visual representation of the grid locations of the subgoals, ignoring the other two state variables. 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 numbers. The mean number of subgoals identified per run was 30.0. Of these, 52% corresponded to picking up the passenger, and 12% corresponded to the navigational subgoals. Another 35% were states within two transitions of these. Figure 2c shows learning curves for LCut, RN, and Q-learning. We used the parameter settings for RN specified in [11], which the authors used for this same task (with identical Q-learning parameters). LCut showed an early improvement in performance in comparison to both RN and Q-learning.

It is important to examine the behaviour of LCut 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 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. We experimented with t_c=0.05, 0.1 and t_p=0.1, 0.3, 0.4, 0.5. The corresponding learning curves are presented in Figure 3. For both settings of t_c, the figure shows a gradual decrement in performance with increasing values of t_p. Interestingly, all settings of the parameters either improved on or replicated the performance of Q-learning.

A promising extention to LCut would be using, instead of a threshold parameter on the proportion of hits, a simple clustering technique (e.g., k-means) that partitions the observed hit proportions into two categories—above and below threshold—without an explicit setting of the threshold.

5 Discussion

Our initial results suggest that LCut is effective in identifying access states working with only recent state trajectories. An alternative use of local cuts is to 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 access states, and would not have to address any issues that result from sampling the graph (as LCut does), but it has the disadvantage of having to build and maintain the entire state transition graph.

LCut is closely related to a number of algorithms proposed in the literature, most notably to QCut [10] and RN [11]. All three algorithms search for the same type of subgoals but differ in how they do this search. 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 of this global graph. In contrast, LCut 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; 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 states visited.

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

[1] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 1043–1049. MIT Press, 1998.

[2] Ronald Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, Computer Science Division, University of California, Berkeley, 1998.

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

[4] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[5] Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

[6] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. MIT Press, 1995.

[7] Marc Pickett and Andrew G. Barto. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In Claude Sammut and Achim G. Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 506–513. Morgan Kaufmann, 2002.

[8] B. Digney. 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, 1998.

[9] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368. Morgan Kaufmann, 2001.

[10] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-Cut - Dynamic discovery of sub-goals in reinforcement learning. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Proceedings of the Thirteenth European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2002.

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

[12] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*. ACM Press, 2004.

[13] R.R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18:534–555, 1999.

[14] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2000.

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

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

[17] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2001.

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