Local Roots: A Tree-Based Subgoal Discovery Method to Accelerate Reinforcement Learning

Alper Demir(✉), Erkin Çilden, and Faruk Polat

Department of Computer Engineering,
Middle East Technical University, 06310 Ankara, Turkey
{ademir,ecilden,polat}@ceng.metu.edu.tr

Abstract. Subgoal discovery in reinforcement learning is an effective way of partitioning a problem domain with large state space. Recent research mainly focuses on automatic identification of such subgoals during learning, making use of state transition information gathered during exploration. Mostly based on the *options framework*, an identified subgoal leads the learning agent to an intermediate region which is known to be useful on the way to goal. In this paper, we propose a novel automatic subgoal discovery method which is based on analysis of predicted shortcut history segments derived from experience, which are then used to generate useful options to speed up learning. Compared to similar existing methods, it performs significantly better in terms of time complexity and usefulness of the subgoals identified, without sacrificing solution quality. The effectiveness of the method is empirically shown via experimentation on various benchmark problems compared to well known subgoal identification methods.

Keywords: Abstraction in reinforcement learning · Subgoal discovery · Options framework

1 Introduction

Subgoal discovery is a prominent way of coping with the scalability problem in reinforcement learning (RL). A subgoal in the problem is a natural hint to partition it into subproblems, so that the agent can focus on learning of smaller tasks, giving rise to opportunities like the transfer of the learned behaviour into a similar problem, and more importantly, increase in learning performance.

Subgoal discovery is almost always coupled with a temporal abstraction mechanism, by which the identified state acts as an artificial target for the problem partition that the agent is trying to solve. A widely accepted temporal abstraction formalism is the *options framework* [23]. An option –which is essentially an abstract action made up of consequent primitive actions through states– defines how to guide the learning agent by making it follow a route to a useful intermediate state, assuming it has the potential to improve learning performance. However, the formalism does not deal with how to decide that an

P. Frasconi et al. (Eds.): ECML PKDD 2016, Part II, LNAI 9852, pp. 361–376, 2016.
DOI: 10.1007/978-3-319-46227-1_23

intermediate state is useful on the way to the ultimate goal. This requirement can effectively be fulfilled by subgoal discovery techniques.

There are a number of different approaches that attack the subgoal discovery problem in RL. Some of the methods are based on graph theory [8,15,19,24], some use statistical methods [3,13,18,20], while others invoke data mining approach [9,11].

Obviously, since the intrinsic focus of RL is on *on-line* performance, it is quite reasonable to expect that the identification of subgoals should better be confluent with the underlying learning procedure. While some methods natively support this paradigm [6,18,19], some others may require additional setup.

In this paper, we propose a subgoal discovery method based on sequence tree based episode history analysis. After each episode, the method first tries to generate a number of successful shortcut policies for every visited state, construct a tree of transitions from generated shortcut policies, and then analyze the tree to extract subgoal states. The method works concurrently with the underlying RL algorithm, and it performs no worse than existing similar methods in terms of solution quality. On the other hand, the method uses less CPU time, and does not depend on any external problem specific variables other than statistical decision parameters. The time complexity of the algorithm is $O((log_b(n))^2)$ on the average, where n is the number of nodes in the generated tree and b is the branching factor. The worst case scenario happens when the agent follows a path through which it visits each state only once causing a tree with branching factor of 1 (which is very unlikely to occur at the initial stages of learning), for which the time complexity is $O(n^2)$.

The paper is organized as follows: A compact summary of the related literature is given in Sect. 2. Section 3 contains the proposed method for subgoal discovery. Experimental evaluation of the proposed algorithm is given in Sect. 4, together with descriptions of problem domains used, parameter settings and a discussion of results. Section 5 includes concluding remarks and possible future research directions.

2 Background

Reinforcement learning (RL) has proven itself to be an effective on-line learning technique [22]. Basically, RL is about self improvements for decisions of a learning agent using environmental feedback. One of the recent advances in RL tries to diminish the diverse effects of increasing state space size, which is known to cause dramatic slow-downs in learning speed. A candidate solution is to partition the problem into manageable pieces and try to solve each first, then ensemble the solutions to obtain the overall result. Lately, subgoal discovery methods have taken attention for this purpose, and are usually coupled with *options framework*.

2.1 Reinforcement Learning

Generally, RL algorithms are constructed on top of a special form of decision process model, called *Markov decision process* (MDP), which possesses *Markov* *property*, meaning that future states of the process depends solely on the current state. With this restricted model, RL algorithms provide a convergence guarantee to the optimal solution under certain conditions, if one exists.

Formally, MDP is a tuple $\langle S, A, T, R\rangle$, consisting of a finite set of *states* S, a finite set of *actions* A, a transition function $T : S \times A \times S \rightarrow [0, 1]$ where $\forall s \in S$, $\forall a \in A$, $\sum_{s' \in S} T(s, a, s') = 1$, and a reward function $R : S \times A \rightarrow \Re$. $T(s, a, s')$ is the probability of being in state s' if action a is performed in state s. $R(s, a)$ gives the immediate reward from the environment after taking action a in state s. A policy $\pi : S \times A \rightarrow [0, 1]$ is a mapping defining the probability of selecting an action in a state. The aim is to find the optimal policy π^* which maximizes the total expected reward received by the agent. If reward and transition functions were known, the optimal policy could easily be found using classical dynamic programming techniques. Otherwise, π^* can effectively be found by estimating the *value function* (i.e. function giving the value of being in a state on the way to goal) incrementally. *Incremental estimation* approach makes use of the average cumulative rewards over different trajectories obtained by following a policy to calculate the value function and gives rise to the central idea of most RL algorithms, called the *temporal difference* (TD) [21].

A famous TD algorithm using action-values (i.e. Q-values) instead of state-values is named Q-Learning [25], and is widely respected due to its simplicity and ease of use. The update rule for Q-Learning is

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a' \in A} Q(s', a')] \tag{1}$$

where $\alpha \in [0, 1)$ is the *learning rate* and $\gamma \in [0, 1)$ is the *discount factor*. Q-Learning has been shown to converge to the optimal action-value function denoted by Q^*, under standard stochastic approximation assumptions.

2.2 Options Framework and Macro-Q Learning

An implicit assumption for the MDP model is that an action lasts for a single time step. However, there are acceptable rationales to relax this assumption. An obvious one would be the convenience in the reuse of a behaviour pattern (i.e. skill) in different situations within the problem space. This *abstraction* idea took attention by various researchers, and a few different mainstream approaches emerged [4,16,23].

A *Semi-Markov Decision Process* (SMDP) extends the MDP model with transitions of stochastic time duration. An SMDP is a tuple $\langle S, A, T, R, F\rangle$, where S, A, T and R define an MDP, and $F(t|s, a)$ is the probability that starting at s, action a completes within time t. MDP is clearly a specialization of SMDP, where a step function has a jump at 1. In the SMDP model, a policy is still a mapping from states to actions, thus the Bellman equations [1] still hold for an optimal policy [2].

As a prominent abstraction formalism based on the SMDP model, *options framework* [23] devices a way to define and invoke timed actions via incorporation of composite actions on top of an MDP model. It allows to create and use abstract actions (*options*) by using primitive actions, lasting for a finite number of discrete time steps. Briefly, an option is defined by three components: (1) a set of states that the option can be initiated at, called the *initiation set*, (2) option's local policy, and (3) a probability distribution induced by the *termination condition*.

A natural extension of Q-Learning to include options is Macro-Q Learning [14], where the value of each primitive action is again updated according to regular Q-Learning (as given in the update rule 1), while the value of an option is updated according to the following rule:

$$\begin{aligned} Q(s_t, o_t) \leftarrow Q(s_t, o_t) + \alpha \times (\gamma^n \times \underset{o'}{max} Q(s_{t+n}, o') - Q(s_t, o_t) \\ + r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n}) \end{aligned} \tag{2}$$

where s_t is the starting state of the option o_t, n is the number of steps taken while the option is employed, s_{t+n} is the state that the option terminates at, o' is the option from s_{t+n} that has the maximal value and r_{t+i} is the reward received at time $t + i$. The reward is discounted by the time it is received.

However, the options framework by itself does not guide or help the designer to grasp some useful abstractions. Thus, automatic generation of those abstractions is another interesting research topic, which has its own variety. A widely used approach is subgoal discovery, where the method seeks bottleneck states or regions in the problem space to derive artificial subgoals to be used as terminating points of the options to be generated.

2.3 Automatic Subgoal Discovery

Automatic discovery of subgoals deals with the problem of identifying a set of intermediate points or regions within an MDP, that are "subgoals" or "bottlenecks", naturally partitioning the problem in hand. Due to the vagueness of the concept, a number of different approaches had been developed for subgoal discovery in RL context.

Some of the methods transform the experience history to a transition graph and analyze it to find most suitable bottleneck regions that partitions the problem [8,15,19,24]. Some other methods rely on state visitation statistics to find frequently used states, based on the observation that frequently visited states are more likely to be a bottleneck on the way to goal [3,7,13,18,20]. A yet different approach interprets the same matter as a clustering problem, trying to find separate regions in state space using experiences and then identify access points between regions as subgoals [9,11]. Although not explicitly subgoal-based, a related family of methods focuses on the sequence analysis on episode histories, under the assumption that the subgoals are signaled via reward peaks [6,12].

However, it is not straightforward to determine whether a state is a subgoal or not. In the ideal case, one needs the complete transition function T in order to make an accurate decision, which is practically not possible. Nevertheless, majority of subgoal discovery methods rely on the assumption that an approximate T can be gathered throughout the RL experience. A drawback of this approach is that it may not be possible to decide that approximation of T is accurate enough to be used for subgoal discovery. Alternatively, few other methods use a hybrid approach that brings together locally collected transition information and a means to statistically test its sufficiency for subgoal discovery [17]. This paper focuses on the latter category of subgoal discovery algorithms.

[17] defines an *access state* as a state that connects two or more connected regions having few transitions in between. The key idea is that, a method searching for an access state is allowed to possess only local statistics throughout the experience, to classify a state as either a target state (an access state) or not. Observations that are collected for a state by this way are then used in the following decision rule:

$$\frac{n_+}{n} > \frac{\ln \frac{1-q}{1-p}}{\ln \frac{p(1-q)}{q(1-p)}} + \frac{1}{n} \frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}} \frac{p(N)}{p(T)})}{\ln \frac{p(1-q)}{q(1-p)}} \tag{3}$$

where n is the total number of observations for a state, n_+ is the total number of positive observations for a state, p is the probability of a positive observation given a target state (an access state), q is the probability of a positive observation given a non-target state, λ_{fa} is the cost of a false alarm, λ_{miss} is the cost of a miss, $p(N)$ is the prior probability of non-target states and $p(T)$ is the prior probability of target states. If the inequality holds, then the state is classified as an access state. This decision rule pinpoints the time step when the collected observations are enough to make a decision about the label of a state. This two-level mechanism enables the methods to avoid the time cost of traversing the whole problem domain.

In the same study, three access state identification methods are proposed. *Relative Novelty (RN)* is a frequency based subgoal identification method, based on the intuition that an *access state* allows the agent to pass from a highly visited region to a new region on the state space. *Local Cuts (L-Cut)* is a graph based method that aims to find a good cut, partitioning the local interaction graph into blocks with a low between-blocks transition probability. *Local Betweenness (LoBet)* is also a graph based algorithm which employs a betweenness measure [5] which is a centrality metric used in graph theory.

Discovered subgoals are of no use unless they are effectively used to diminish the adverse effects of the large state space. Usually, options framework is used to achieve this purpose. For each subgoal identified, an option towards that state is generated. The initiation set for the option is formed by adding the states observed before the subgoal in each trajectory.

One common way of generating the policy of an option to reach a subgoal is the *Experience Replay (ER)* mechanism [10]. ER reuses the past experiences of the agent to find a policy to reach the identified subgoal by providing artificial rewards rather than the actual reward yielded by the environment. A general convention is to provide a positive reward upon reaching the subgoal state, and a negative reward for any other transition.

Algorithm 1. *LOCAL_ROOTS*

```
Require: $p$, $q$, $\frac{\lambda_{fa}}{\lambda_{miss}}$, $\frac{p(N)}{p(T)}$
 1: $o_i \leftarrow 0$, $o_i^+ \leftarrow 0$
 2: for each episode do
 3:     $h \leftarrow$ Interact with the environment                ▷ record episode history
 4:     if $h$ ended with a peak reward then
 5:         $nt_{avg} \leftarrow$ calculate average number of distinct transitions in $h$
 6:         $T \leftarrow CREATE\_TREE(h)$
 7:         $CALCULATE\_ROOTING\_FACTORS(T, nt_{avg})$              ▷ for each vertex
 8:         for $s \in V_T$ do
 9:             $o_s \leftarrow o_s + 1$
10:             if s is a local maximum on $T$ then
11:                 $o_s^+ \leftarrow o_s^+ + 1$
12:             end if
13:             if the decision rule is satisfied then         ▷ use Decision Rule 3
14:                 Classify $s$ as a subgoal
15:             end if
16:         end for
17:     end if
18: end for
```

3 Local Roots Method for Subgoal Discovery

We define a subgoal as a state that serves as a junction point or a region of the known shortcut paths from each state to the goal state, which is in fact a likely bottleneck candidate. Following this intuitive definition, our approach depends on the notion of a successful trajectory, that is, a trajectory ending with a distinctive reward peak, which is usually the goal state of the problem domain.

Our method named *Local Roots* generates positive and negative observations for visited states and feeds them to the Decision Rule 3. This is a common pattern in local approaches in subgoal discovery, since the local information gathered from the episode history can be highly dependent on the particular way of state visitations, and thus, may give rise to noisy results without a high level decision filter (especially false positives). Decision Rule 3 is calculated for each visited state, aiming to distinguish the subgoals more accurately.

Local Roots method records the transition history for each episode (Algorithm 1, line 3). Upon completion of an episode, it first checks whether the last transition yields the maximum reward for that episode, or not. If so, it calculates the average number of distinct transitions made through a state (nt_{avg}) and creates a tree using shortcut paths derived using state equivalences, to serve as a collection of the best "memorized" trajectories starting from every visited state up to the goal state (Algorithm 1, line 6). Best trajectories are calculated by traversing from a leaf of the tree to the root, iteratively updating the transitions with the best values. The path from a vertex to the root in the tree forms the shortest path from the corresponding state to the last state in the episode. The tree generation procedure is given in Algorithm 2. The resulting

Algorithm 2. *CREATE_TREE*

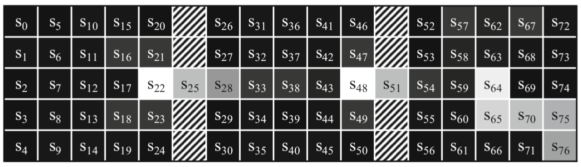
```
Require: a successful episode trajectory $h$
Ensure: a tree $T$ representing shortcut histories to goal from each state
 1: $t \leftarrow length(h) - 2$
 2: $V_{s_{t+1}} \leftarrow 0$, $best_{s_{t+1}} \leftarrow null$
 3: while $t \geq 0$ do
 4:     if $V_{s_t}$ is undefined $\vee$ $(r_{t+1} + \gamma * V_{s_{t+1}}) > V_{s_t}$ then
 5:         $V_{s_t} \leftarrow r_{t+1} + (\gamma * V_{s_{t+1}})$
 6:         $best_{s_t} \leftarrow s_{t+1}$
 7:     end if
 8:     $t \leftarrow t - 1$
 9: end while
10: $V \leftarrow \{s_l\}, E \leftarrow \emptyset$
11: for each state $s \neq s_l$ do
12:     $V \leftarrow V \cup \{s\}$
13:     $E \leftarrow E \cup (s, best_s)$
14: end for
15: return $(V, E)$
```

tree is a collection of shortcut paths (i.e. free of loops) from every visited state to the goal state, based on the local transition graph derived from experiences.

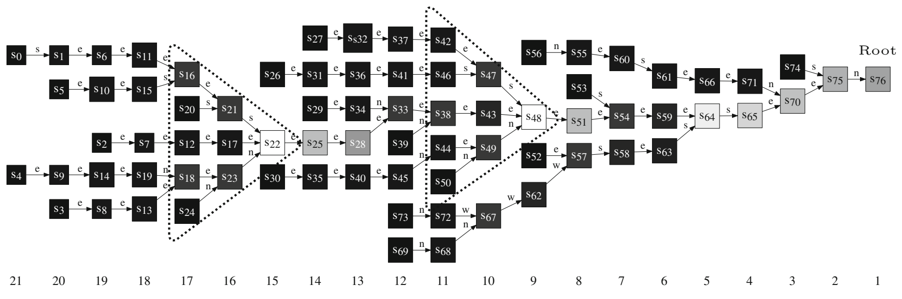
The core idea of the proposed method lies in a state metric, what we call the *rooting factor*, due to the visual resemblance to a root structure of a tree in the nature fringing underground. To clarify the idea, Fig. 1a illustrates a sample grid world domain made up of three rooms with passageways between adjacent rooms, and Fig. 1b is a tree generated for the problem by using an episode history. The goal state is s_{76} which is located in the south-east corner of the room and the agent starts from the north-west corner, namely s_0. The agent can move to any one of the four compass directions at each time step, except that after a move attempt to the walls and the boundaries of the room, it stays still.

The grid world and the tree instance given in Fig. 1 are colored according to the rooting factor values of states scaled from black to white, where brighter color means a higher value. Note that the states in the doorways have high rooting factor values. In the case of state s_{48} located at level 9, the rooting factor metric focuses on the sub-tree having state s_{48} as the root. To calculate the rooting factor of state s_{48}, one should first find the *widest level* for the sub-tree rooted by state s_{48}, which is the level possessing the first peak in sub-tree width. In this sub-tree, the widths of each level are 1, 3, 5, 5, 5, 5, 3 consecutively, and the first peak value in terms of width is 5. The method considers level 11 as the widest level and ignores the second peak with value 5 starting in level 17 since level 11 has the first peak. This way, even if there is a wider level below, it is not taken into account for state s_{48}. Thus, each bottleneck state in the tree possesses its relative sub-tree, as is the case for another subgoal s_{22} in level 15 where the widest level of its relative sub-tree is level 17.

Upon identification of the widest level for state s_{48}, one can think of an imaginary triangle (i.e. the dotted triangle in Fig. 1b) where the vertices in the



(a)



(b)

Fig. 1. (a) A sample grid world with two consecutive subgoals, colored according to rooting factor values of the states. Shaded cells represent walls. (b) The generated tree, using the same coloring scheme. Actions are noted on the edges. The numbers at the bottom are corresponding levels of the tree.

widest level compose its base, and its topmost corner is s_{48} (w.r.t. a portrait orientation of the tree, where root is at the top). The shape of this triangle is an indication of the "importance" of state s_{48} in the tree. A wider triangle suggests that, for relatively more states, the agent should pass through state s_{48} in order to reach the root state (i.e. goal). The height of the triangle, on the other hand, pinpoints a state which is the "first" junction point of the merging paths. That is why, the rooting factor of state s_{48} is higher than its parent's, state s_{51}.

As a mathematical interpretation of the above characteristics, the rooting factor of s can be defined as follows:

$$r_s = \frac{(n_{widest})^{nt_{avg}}}{d_{widest} - d_s} \tag{4}$$

where d_s is the depth of s in the tree, nt_{avg} is the average number of distinct transitions of states in the tree, n_{widest} is the number of vertices in the widest level and d_{widest} is the depth of the states in that level. In order to strengthen the effect of possible connections that a vertex can have, number of vertices in the widest level is powered by the average number of distinct transitions (nt_{avg}).

After the tree is constructed by using Algorithm 2, the rooting factor calculation takes place for every vertex (Algorithm 1, line 7). Algorithm 3 is employed for this purpose, where a tree traversal (via breadth first search, BFS) is employed first, to find the depth of each vertex in the tree. An addi-

Algorithm 3. *CALCULATE_ROOTING_FACTORS*

```
Require: a successful history tree $T$, $nt_{avg}$
 1: calculate the depth of each state in $T$                        ▷ use BFS
 2: $d_{max} \leftarrow \max_{s \in V_T}(\text{depth}(s))$                 ▷ find maximum depth
 3: for every $s \in V_T$ do
 4:     $n_i(s) \leftarrow$ number of nodes at depth $i \geq \text{depth}(s)$ in the subtree rooted at $s$
 5: end for
 6: for each state $s$ in $V_T$ do          ▷ rooting factor calculation for every vertex
 7:     $d_s \leftarrow \text{depth}(s)$
 8:     $i \leftarrow d_s, n_{widest} \leftarrow 1, d_{widest} \leftarrow d_s$
 9:     while $i \leq d_{max}$ and $n_i(s) \geq n_{widest}$ do
10:         if $n_i(s) > n_{widest}$ then
11:             $n_{widest} \leftarrow n_i(s)$
12:             $d_{widest} \leftarrow i$
13:         end if
14:         $i \leftarrow i + 1$
15:     end while
16:     $r_s \leftarrow$ calculate the rooting factor of $s$ using Eq. 4
17: end for
```

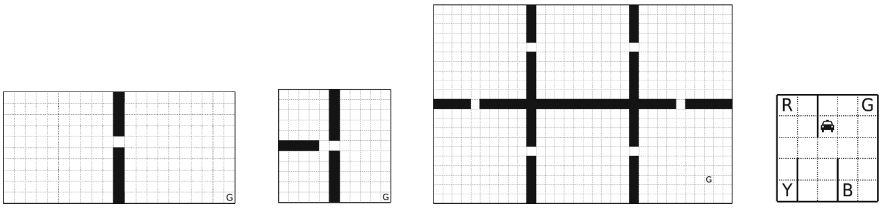
tional traversal is run afterwards, from the level with the deepest state(s) to the root, to find the number of vertices below each vertex classified by their depths (Algorithm 3, lines 3–7). Using this information, the rooting factor of each state can be calculated by traversing from the state under consideration to the deeper levels.

Having calculated the rooting factor values for every visited state, each state is checked whether it is a local maximum or not, in terms of rooting factors, among its children and parent in the tree. The state gets a positive observation if that is the case, or a negative observation otherwise. The root of the entire tree does not get a positive observation since it is a possible goal state and is obviously not a subgoal. The observations made for each state are fed to the Decision Rule 3 for a further classification.

The most time consuming portion of the Local Roots algorithm is the part where the rooting factor value is calculated for each state which has $O(n^2)$ worst time complexity. However, the worst case happens when the tree is linear (although it is usually unlikely at the initial states of learning due to the exploration component, the agent might execute a policy that visits each state only once) and the branch factor (b) is 1, which means, by our definition, there is no new subgoal to identify. Thus, the worst case can be avoided by a heuristic check on the shape of the generated tree. On the other hand, the algorithm performs $O((\log_b(n))^2)$ on the average, where n is the number of nodes in the tree. Since the algorithm makes use of local episode trajectories, the number of nodes in the tree (n) does not directly relate to the number of states in the whole domain.

4 Experiments

We tested the algorithms on four grid world navigation domains (Fig. 2), three of the which are well known benchmark problems in the related literature. State and action set sizes of problems and corresponding references are given in Table 1. A new 3 rooms grid world problem (Fig. 2b) is designed to investigate the subgoal identification behaviour of the methods in a 3-way junction situation. Local Roots, just like the other similar methods, transforms the problem to a transition graph. Thus, the method is essentially independent of domain specific structure. Our motivation for experimenting on grid world domains is to better visualize the bottleneck idea for the reader.



(a) 2 rooms (b) 3 rooms (c) 6 rooms (d) Taxi

Fig. 2. Problem domains

In 2, 3 and 6 rooms problems, the agent can perform four movement actions, which are *north*, *east*, *south* and *west*. The environment is non-deterministic, since the agent moves to the intended direction with probability of 0.9 and moves randomly in any of the movement directions with 0.1 probability. The reward for reaching the goal state G is 1.0 while the reward for any other transition is 0. In the 2 and 3 rooms problems, the agent starts from any cell in the left room(s) while it starts from any cell in the upper left room in the 6 rooms domain.

The last problem is the famous Taxi domain (Fig. 2d, [4]), in which a taxi tries to pick a passenger from its location and transfer it to a destination location in a 5×5 grid world with designated locations. The taxi agent can perform 6 actions: movement actions *north*, *east*, *south*, *west*; a *pickup* action to get the passenger, and a *putdown* action to drop the passenger. The passenger is initially located in 4 different cells marked as R, Y, G and B, and the destination of the passenger is one of these four designated cells. The actions are noisy, leading the agent to its intended direction with probability of 0.8 and randomly moving it to the left or the right of the intended direction with probability of 0.1 each. The agent is punished for wrong pickups and putdowns with -10 and it is rewarded with $+20$ when it puts down its passenger in the desired location. Any other transition is given the reward of -1.

4.1 Settings

We compared Local Roots method (LoRoots) with RN, LoBet and L-Cut, since they use the same decision rule and can be employed on-line. The decision rule

Table 1. Problem sizes and parameter values used

Problem	Size		Parameters used							Ref.
	$\|S\|$	$\|A\|$	Method	p	q	t_c	t_{RN}	k	l_n	
2 rooms	201	4	RN	0.06	0.01	-	2.0	2	7	[19]
			L-Cut	0.3	0.01	0.05	-	-	-	
			LoBet	0.7	0.07	-	-	-	-	
			LoRoots	0.6	0.06	-	-	-	-	
3 rooms	106	4	RN	0.05	0.01	-	2.0	2	7	-
			L-Cut	0.1	0.01	0.05	-	-	-	
			LoBet	0.6	0.06	-	-	-	-	
			LoRoots	0.6	0.06	-	-	-	-	
6 rooms	605	4	RN	0.5	0.008	-	2.0	2	7	[15]
			L-Cut	0.2	0.01	0.05	-	-	-	
			LoBet	0.5	0.05	-	-	-	-	
			LoRoots	0.75	0.05	-	-	-	-	
Taxi	500	6	RN	0.712	0.01	-	2.0	2	7	[4]
			L-Cut	0.04	0.002	0.05	-	-	-	
			LoBet	0.3	0.03	-	-	-	-	
			LoRoots	0.24	0.03	-	-	-	-	

parameters were optimized separately for each method and problem so that they find subgoals in the early stages of learning and they eliminate noise properly. The cost ratio ($\lambda_{fa}/\lambda_{miss}$) and the prior ratio ($p(N)/p(T)$) parameters of Decision Rule 3 were set to 100 for all experiments. The visitation counts used by RN were reset at the end of each episode. The remaining parameters used by the subgoal identification methods are given in Table 1. Unfortunately, there is no practical way to find the correct values other than a number of trial-and-error experimentation sessions. Specifically, a heuristic we used to set p and q values is, to examine the outputs of the used subgoal discovery methods and calibrate them according to the subgoals that we manually identified. Other parameters are mostly inherited from [17] where a further analysis can be found.

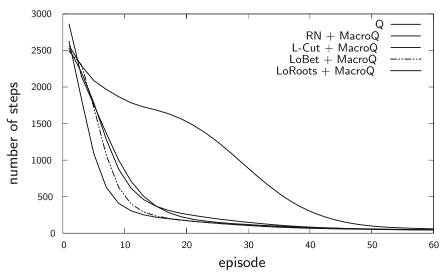
When a subgoal is found, the agent generates an option to reach that subgoal. The initiation set of the new option contained the states before the first occurrence of the subgoal in each previous episode. *Option lag* (l_o), the number of time steps to look for states to add the initiation set, was 10. Termination probability for each state in the initiation set was set to 0.0, while 1.0 was used for the subgoal. The policy of the option was formed through ER by giving 100 reward upon reaching the subgoal, -10 punishment for leaving the initiation set and -1 punishment for any other transition. For the policy learning part of ER, $\alpha = 0.125$ and $\gamma = 0.9$ were used as the learning parameters. The replay is repeated 10 times for fast convergence.

The agent incorporated Macro-Q learning algorithm, where Q values of an option were updated according to Macro-Q learning while Q values of primitive actions were updated according to regular Q learning. ϵ-greedy was used as option selection strategy, with $\epsilon = 0.1$, and $\alpha = 0.05$ and $\gamma = 0.9$ were set as learning parameters. The same γ value is used in Algorithm 2. All of the results are averaged over 200 experiments.

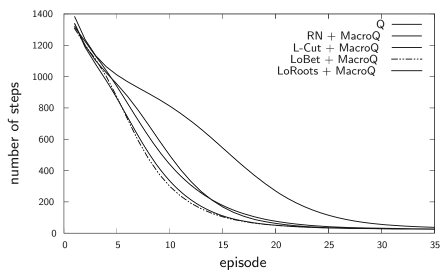
4.2 Results and Discussion

Average number of steps to reach the goal state are compared among methods, and the results are sketched in Fig. 3. Plots are smoothed for visual clarity. All of the subgoal identification methods, including Local Roots, improve the learning speed of the agent by leading it to the goal state earlier and our proposed method matches the performance of the other methods. In general, subgoals discovered by Local Roots seem to be as useful as the ones found by L-Cut, LoBet and RN. We can conclude that the solution quality of Local Roots method is not worse than others on the average.

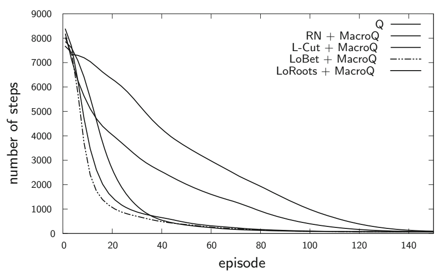
LoBet algorithm has $O(n \cdot m)$ and $O(n \cdot m + n^2 \cdot logn)$ time complexities on unweighted and weighted graphs respectively, while L-Cut algorithm requires $O(n^3)$ time when the local interaction graph has n vertices and m edges. On the other hand, RN algorithm is $O(1)$. The time complexity of our proposed method, Local Roots, depends on the maximum depth of a state in the tree it creates.



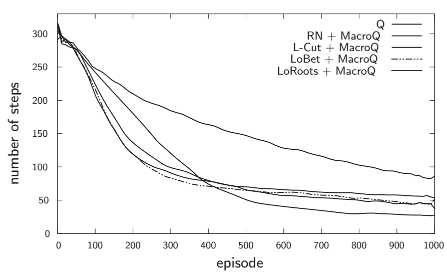
(a) 2 rooms



(b) 3 rooms



(c) 6 rooms



(d) Taxi

Fig. 3. Average number of steps to goal for each problem

It requires $O((\log_b(n))^2)$ time where b is the average branching factor and n is the number of vertices in the tree.

CPU time measurements, indicating the CPU times used by each subgoal discovery method excluding the underlying Macro-Q algorithm, are given in Table 2. The results shows that both LoBet and L-Cut require much more time than Local Roots because of their higher time complexities. The only exception is LoBet for Taxi problem, whose time consumption seems to be nearly the same as in Local Roots. On the other hand, although RN is $O(1)$, its time complexity is in fact associated with the number of steps taken, since it is invoked at every time step, unlike the other methods waiting for the episode end. Longer episodes in the earlier stages of an experiment causes RN to generally take more time than Local Roots. Table 2 implies that Local Roots algorithm shows a significant advantage in terms of CPU time compared to other methods.

Table 2. Average CPU time overhead per episode (msec)

Problem	RN	L-Cut	LoBet	LoRoots
2 rooms	0.63	5.18	0.65	0.45
3 rooms	0.33	1.78	0.32	0.24
6 rooms	11.00	289.30	7.10	4.08
Taxi	0.85	1.68	0.79	0.81

Figure 6 shows the subgoals discovered by all four methods for 2 rooms problem, marked with brighter color showing high frequency of identification. L-Cut, LoBet and RN finds more than one subgoals including the doorway and states one step near to it. On the other hand, our proposed method finds only the state before the doorway as it is the first merging point of the shortest paths of the states in the left room to the goal state in the right room. This characteristic causes Local Roots to find less number of subgoals than the other algorithms, especially in 2, 3 and 6 rooms domains. However, as seen Fig. 4, effectiveness of subgoals discovered are usually higher in Local Roots method compared to the others. We define the *effectiveness of a subgoal* as the $(100 \times n_{steps(option)}/n_{steps(episode)})/n_{subgoals}$, where $n_{steps(option)}$ is the total number steps passed within option sequences, $n_{steps(episode)}$ is the total number of steps taken during the episode, and $n_{subgoals}$ is the number of subgoals identified at the end of an episode. Subgoal effectiveness can be interpreted as the ability of a subgoal to trigger a useful option. Contribution of some of the additional subgoals found by the other three methods are not as significant as that are found by Local Roots in general.

Finally, average memory usage of Local Roots does not exceed the graph based methods (i.e. LoBet and L-Cut) in general, since it uses a tree instead of a graph, having less number of edges than the graphs used by the other methods. It is worth noting that, memory usage metrics also include ER repositories (Fig. 5).

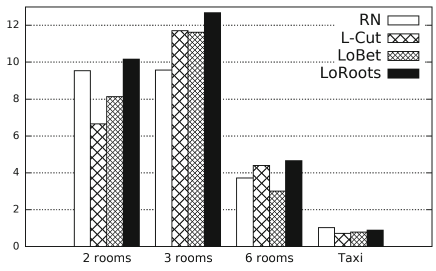


Fig. 4. Average subgoal effectiveness (average option trace % per subgoal)

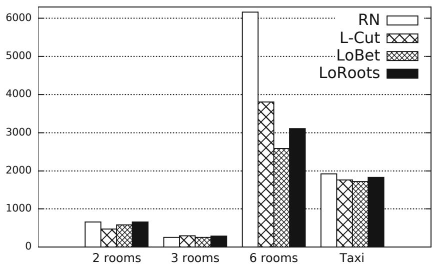


Fig. 5. Average memory usage per episode in kilobytes

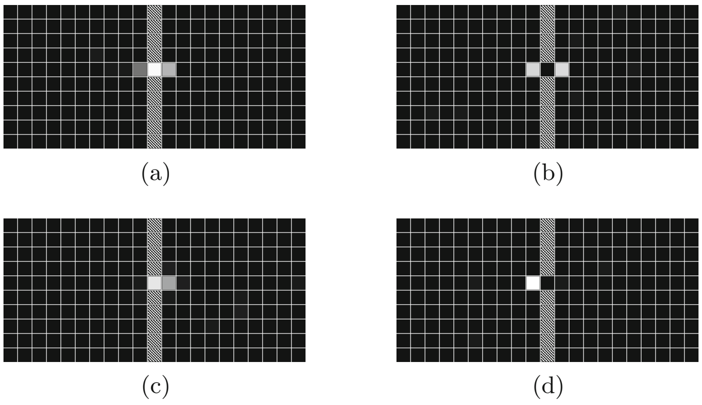


Fig. 6. Subgoals found in 2 rooms domain by (a) L-Cut (b) LoBet (c) RN (d) Local Roots.

In addition to the parameters of the Decision Rule 3, L-Cut requires one and RN requires four more parameters while LoBet and Local Roots require none. These extra parameters determine the quality of the subgoals found by L-Cut and RN and make them dependent on the structure of the domain. In that sense, Local Roots, like LoBet, is less dependent on the problem characteristics compared to L-Cut and RN. Moreover, as seen in Fig. 4, Local Roots outperforms LoBet in terms of subgoal quality.

5 Conclusion

In this paper, we propose a tree based automatic subgoal discovery method called Local Roots that helps the learning agent to identify important states on the way to the goal state in the early stages of learning. Local Roots method can be employed upon reward peaks, which are usually goal states. Using the options framework, the learning agent can devise abstractions to reach the identified subgoals. The method utilizes a tree based metric to locally identify the junction points of the shortcuts directed from each visited state towards the goal state.

In terms of learning speed, Local Roots outperforms the regular Q-Learning for all problem domains experimented. It also keeps up with the performance of the other local methods on the average, showing that subgoals identified by Local Roots are no worse than the ones found by other algorithms.

Compared to other graph based methods tested, Local Roots has lower time complexity. On the other hand, when average CPU times per episode are compared, Local Roots outperforms all other methods on the average, including Relative Novelty which has the lowest theoretical time complexity, but should be invoked at every time step. Local Roots is also shown to identify less number of subgoals with higher effectiveness in general. Moreover, it requires no additional parameters unlike Relative Novely and Local Cuts.

A possible future research direction is to find an alternative way of discriminating noise from local subgoal information with less domain specific parameters. Also, automatic detection of these parameters can be an important improvement for all the online methods presented here.

Acknowledgments. This work is partially supported by the Scientific and Technological Research Council of Turkey under Grant No. 215E250.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
2. Bradtke, S.J., Duff, M.O.: Reinforcement learning methods for continuous-time markov decision problems. In: Tesauro, G., Touretzky, D., Leen, T. (eds.) Advances in Neural Information Processing Systems, NIPS 1994, vol. 7, pp. 393–400. MIT Press, Cambridge (1994)
3. Chen, F., Chen, S., Gao, Y., Ma, Z.: Connect-based subgoal discovery for options in hierarchical reinforcement learning. In: Lei, J., Yao, J., Zhang, Q. (eds.) Proceedings of the Third International Conference on Natural Computation, ICNC 2007, vol. 4, pp. 698–702. IEEE (2007)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. **13**(1), 227–303 (2000)
5. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**(1), 35–41 (1977)
6. Girgin, S., Polat, F., Alhajj, R.: Improving reinforcement learning by using sequence trees. Mach. Learn. **81**(3), 283–331 (2010)
7. Goel, S., Huber, M.: Subgoal discovery for hierarchical reinforcement learning using learned policies. In: Russell, I., Haller, S.M. (eds.) Proceedings of the 16th International FLAIRS Conference, pp. 346–350. AAAI Press (2003)
8. Kazemitabar, S.J., Beigy, H.: Automatic discovery of subgoals in reinforcement learning using strongly connected components. In: Köppen, M., Kasabov, N., Coghill, G. (eds.) ICONIP 2008, Part I. LNCS, vol. 5506, pp. 829–834. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02490-0_101
9. Kheradmandian, G., Rahmati, M.: Automatic abstraction in reinforcement learning using data mining techniques. Robot. Auton. Syst. **57**(11), 1119–1128 (2009)
10. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. **8**(3), 293–321 (1992)

11. Mannor, S., Menache, I., Hoze, A., Klein, U.: Dynamic abstraction in reinforcement learning via clustering. In: Proceedings of the Twenty-first International Conference on Machine Learning, ICML 2004, pp. 71–78. ACM (2004)
12. McGovern, A.: acQuire-macros: an algorithm for automatically learning macro-actions. In: The Neural Information Processing Systems Conference Workshop on Abstraction and Hierarchy in Reinforcement Learning, NIPS 1998 (1998)
13. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML 2001, pp. 361–368. Morgan Kaufmann Publishers Inc. (2001)
14. McGovern, A., Sutton, R.S., Fagg, A.H.: Roles of macro-actions in accelerating reinforcement learning. In: Proceedings of the 1997 Grace Hopper Celebration of Women in Computing, pp. 13–18 (1997)
15. Menache, I., Mannor, S., Shimkin, N.: Q-Cut–Dynamic discovery of sub-goals in reinforcement learning. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) ECML 2002. LNCS, vol. 2430, pp. 295–306. Springer, Heidelberg (2002)
16. Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Jordan, M., Kearns, M., Solla, S. (eds.) Advances in Neural Information Processing Systems, NIPS 1997, vol. 10, pp. 1043–1049. MIT Press (1998)
17. Simsek, O.: Behavioral building blocks for autonomous agents: description, identification, and learning. Ph.d. thesis, University of Massachusetts Amherst (2008)
18. Simsek, O., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004, pp. 95–102. ACM (2004)
19. Simsek, O., Wolfe, A.P., Barto, A.G.: Identifying useful subgoals in reinforcement learning by local graph partitioning. In: Proceedings of the Twenty-second International Conference on Machine Learning, ICML 2005, pp. 816–823. ACM (2005)
20. Stolle, M., Precup, D.: Learning options in reinforcement learning. In: Koenig, S., Holte, R.C. (eds.) SARA 2002. LNCS, vol. 2371, pp. 212–223. Springer, Heidelberg (2002)
21. Sutton, R.S.: Learning to predict by the methods of temporal differences. Mach. Learn. **3**(1), 9–44 (1988)
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning. MIT Press, Cambridge (1998)
23. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)
24. Taghizadeh, N., Beigy, H.: A novel graphical approach to automatic abstraction in reinforcement learning. Robot. Auton. Syst. **61**(8), 821–835 (2013)
25. Watkins, C.: Learning from delayed rewards. Ph.d. thesis, Cambridge University (1989)