Automatic Abstraction in Reinforcement Learning Using Ant System Algorithm

Mohsen Ghafoorian, Nasrin Taghizadeh and Hamid Beigy
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
{ghafoorian, taghizadeh}@ce.sharif.edu, beigy@sharif.edu

Abstract

Nowadays developing autonomous systems, which can act in various environments and interactively perform their assigned tasks, are intensively desirable. These systems would be ready to be applied in different fields such as medicine, controller robots and social life. Reinforcement learning is an attractive area of machine learning which addresses these concerns. In large scales, learning performance of an agent can be improved by using hierarchical Reinforcement Learning techniques and temporary extended actions. The higher level of abstraction helps the learning agent approach lifelong learning goals. In this paper a new method is presented for discovering subgoal states and constructing useful skills. The method utilizes Ant System optimization algorithm to identify bottleneck edges, which act like bridges between different connected areas of the problem space. Using discovered subgoals, the agent creates temporal abstractions, which enable it to explore more effectively. Experimental Results show that the proposed method can significantly improve the learning performance of the agent.

Introduction

Reinforcement Learning (RL) is an active area of the machine learning which considers the problem of how an intelligent agent can learn an optimal behavior through trial-and-error with a stochastic dynamic environment. A challenging problem in RL is how to scale up standard methods for large complex tasks. Researches on the problem of reducing search space and transferring knowledge across problems led to introduction of the hierarchical reinforcement learning frameworks (HRL) (Parr and Russell 1998; Sutton, Precup, and Singh 1999; Dietterich 2000).

In most applications, hierarchical structure of the problem is defined by the system designer prior to learning process. However, it becomes more difficult complex task when the size of state space increases; hence, it is desirable to minimize designer's role and to construct hierarchies by the learning agent automatically, as in many cases it is not so straightforward. Most of the researches on this topic have focused on identifying possible subgoals of the problem and learning policies to reach them.

Several definitions have been proposed to describe subgoals: states which are visited frequently or have a high reward gradient (Digney 1998), states which are visited frequently on successful trajectories but not on unsuccessful ones (McGovern 2002), states which lie between densely-connected regions of the state space (Menache, Mannor, and Shimkin 2002; Mannor et al. 2004; Şimşek 2008), states which lead to a new area of the state space (Şimşek 2008), states which connect dense regions of the state space and the transition probability from one region to the other is low (Kazemitabar and Beigy 2008). Using identified subgoals, the agent can discover temporally abstracted actions to create hierarchy of the solutions: Q-cut method uses max-flow/min-cut algorithm (Menache, Mannor, and Shimkin 2002). Strongly connected component method (SCC) (Kazemitabar and Beigy 2009) employs depth-first search (DFS) algorithm. The concept of relative novelty is used in the method presented at (Şimşek 2008). Reformed label propagation utilizes modularity measure (Davoodabadi and Beigy 2011); and value based clustering algorithm uses reward information (Kheradmandian and Rahmati 2009), to mention a few.

In this paper, we propose a new method for temporal abstraction in RL based on identifying subgoal states. In our definition, subgoals are states adjacent with those edges which are visited regularly on the shortest paths between start and goal states (Ghafoorian 2012). In order to measure how much a node is visited regularly, we propose to use *roughness* criterion. One promising approach in subgoal discovery is to use information acquired by the agent during learning process. In the proposed method, the agent begins to explore the environment and saves history of interactions in a directed and weighted graph. Next, the agent uses Ant System algorithm in order to find the shortest path between start and goal states. Analyzing pattern of changing in pheromone of the edges placed on the shortest path can help to specify those edges that act like a bridge between strongly connected areas of the state space. We call these edges *bottleneck edges*. Then desired options will be created to reach these subgoals and will be added to the agent's available action set. We demonstrate the effectiveness of the proposed method using computer simulations. The results of simulations show that the proposed method attains substantial improvement in learning speed of the agent.

This paper is organized as follow: In section 2, we explain the standard RL framework and option notion. In section 3, Ant System method and motivation of choosing this algorithm for discovering subgoals will be described. Next, we introduce our skill acquisition algorithm based on Ant System. Experimental results are reported in sections 4, section 5 contains discussion and finally, conclusion and future works are presented at section 6.

Background

The standard model of RL is discrete time MDP with a finite set of states, S and a finite set of actions, A (Sutton, Precup, and Singh 1999). At each time step $t, t = 1, 2, \ldots$, the agent knows the state $s_t \in S$ and chooses an action a_t from available actions at state s_t, $A(s_t)$. Then the agent observes next state, s_{t+1}, and receives a scalar reward, r_t. The agent's task is to find a mapping from states to actions, called policy, $\pi : S \rightarrow A$, which maximizes the expected discounted cumulative reward: $E\{\sum_{t=0}^{\infty} \gamma^t r_t\}$, where $0 \leq \gamma \leq 1$ is the discount factor.

In the well-known Q-learning algorithm, the agent maintains an estimation of the optimal Q function and updates it after every time epoch. This function maps each state-action pairs to the expected reward for taking this action at that state and following an optimal strategy from that point on (Mannor et al. 2004). An extension of Q-learning is Macro-Q-learning (or Q-learning with options). Option is a triple (I, π, β), where I is initial set, π is option policy and β is termination condition. Option can start in every state of I; then it is executed according π until the termination condition is satisfied which is denoted by $\beta(s)$. When the agent is not executing an option, it can choose a primitive action or an option from $A'(s_t)$, which is the set of all actions and options available at s_t. Macro-Q-learning updates Q-function as equation (1):

$$Q(s_t, o_t) = (1-\alpha)Q(s_t, o_t) + \alpha\left(\gamma^\tau \max Q(s_{t+\tau}, o_t) + \sum_{k=0}^{\tau-1} r_{t+k}\gamma^k\right), \quad (1)$$

where τ is duration of option o_t, α is learning rate. If $\tau = 1$, updating rule for primitive actions is obtained.

The Proposed Skill acquisition Method

Most of methods which are based on temporal abstraction use sub-goal discovery in order to create options. In most cases, sub-goals are defined as the border states of the strongly connected areas, emerged in the structure of transition graph. This way, several skills will be generated, while some of them may be irrelevant for the task in hand, and thus all of skills are not necessarily useful for the agent to reach the goal. Furthermore, selecting these useless options will mislead the agent in many cases. For example, standing up may be considered as a skill for a human, but it may not be useful when a person is writing an article, and also it is a misleading option for the task of writing an article. Some algorithms try to filter generated options in a post processing phase (Taghizadeh 2011). However, it may be faulty and time consuming. In this paper, we propose a method, which avoid generating any of the possible useless options and will identify and use only the effective options. In addition, many algorithms do not make use of all the information taken from the environment such as transitions direction between states and frequency of those transitions. These data are also considered as important directions in the proposed method. The agent creates a directed and weighted graph $G = (V, E, W)$ using history of its interactions with the environment, such that every visited state is a vertex $v \in V$, and every transition is an edge $e \in E$ in the graph. The weight of each edge, $w_{ij} \in W$, is the inverse of number of transitions through this edge. The method presented in this paper is inspired by Ant Colony optimization method, which will be discussed in next subsection.

Ant Colony Optimization

Ant colony optimization (Dorigo, Birattari, and Stutzle 2006) is a family of evolutionary methods, which can be used in order to find the shortest path between a pair of nodes in a graph $G = (V, E)$. There are plenty of methods in this family of algorithms. In this paper we will use *Ant System*.

Ant System consists of n_t iterations. In each iteration, there are n_k ants and each of them generates a path from start node s to the destination node t. Generation of each path is a random process guided by the amount of pheromone deposited by previous ants on the edges of their path. Formally, when ant k-th is on node i, the probability for selecting j as the next node to traverse, may be calculated by:

$$p_{ij}^k(t) = \begin{cases} \frac{\alpha\tau_{ij}(t)+(1-\alpha)\eta_{ij}(t)}{\sum_{j\in N_j^k(t)} \alpha\tau_{ij}(t)+(1-\alpha)\eta_{ij}(t)} & \text{if } j \in N_i^k(t) \\ 0 & \text{if } j \notin N_i^k(t) \end{cases} \quad (2)$$

where N_i^k is the set of all accessible nodes from node i for k-th ant, τ_{ij} is the amount of pheromone on edge (i, j), η_{ij} is a heuristic that indicates the prior knowledge about desirability of transition from i to j and finally, α is a real number in range $(0, 1)$ that represents the relative importance of pheromone values. It worth being noticed that a tabu list is kept, each time creating the path so as to prevent the creation of loopy paths.

After all n_k ants create their own path to destination node, evaporation takes place in order to prevent trapping in local optima, using the following equation:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t), \quad (3)$$

where $\rho \in (0, 1)$. This constant specifies the rate at which the pheromone evaporates.

After evaporation, all n_k ants deposit pheromone on edges on the path they created. The amount of pheromone deposited by k-th ant, on link (i, j) is given by:

$$\Delta\tau_{ij}^k(t) \propto \frac{1}{L^k(t)}, \quad (4)$$

where $L^k(t)$ is the length of path constructed by k-th ant, at time step t. Amount of pheromone on link (i, j) is updated

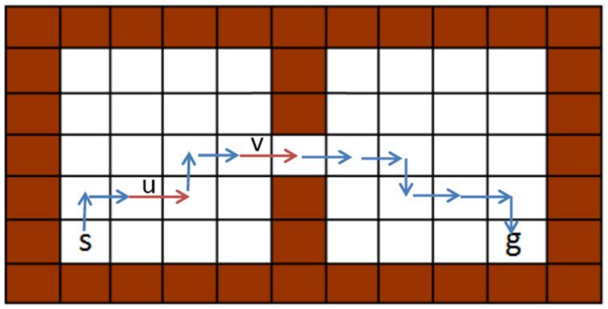


Figure 1: A shortest path between s and g found by Ant System

using following equation:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^{k}(t). \tag{5}$$

Using this process iteratively, edges on the shorter paths are rewarded getting more amount of pheromone and thus they get higher probability of being chosen in the next paths. Termination criterion can be selected among different options. One may use an upper bound for number of iteration, or terminate when the quality of generated paths exceeds a limit.

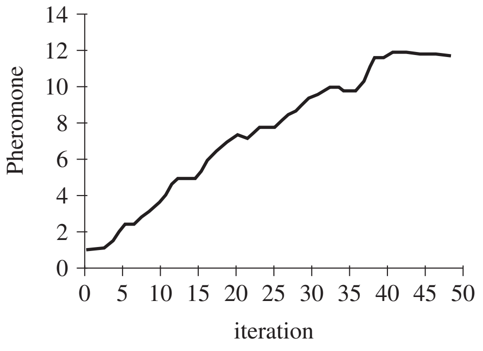
Sub-goal Discovery

According to the agent's task in hand, some of options may be useless. Considering the fact that we are going to avoid these useless options, the task being fulfilled must be regarded. A task is defined by its initial and final states in the environment. In many environments these particular states are defined as an environment property rather than task. For example in chess playing environment, initial state is task independent. Finally, we assume that initial state s and final state t are known.

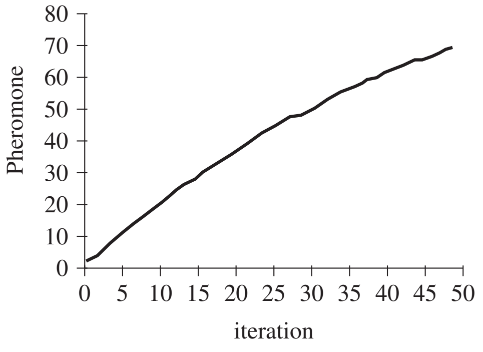
Considering the simple grid world environment, which is depicted in Figure 1, the task for the agent is to reach the goal states g, starting form starting state s. Clearly in this environment, the door is a subgoal and there exist two rational skills, one helps the agent reach the door, and the other helps it approach goal from door.

When Ant System method is applied to the transition graph of this environment, after several iterations of creating paths from start to goal node, a shortest path to the destination node will be found as shown in Figure 1. In transition graphs, which have apparent community structure, there are some edges on the shortest path, which act like a bridge between adjacent communities. We call these edges *bottleneck edges*. The main step of the skill discovery algorithm is to identify these bottleneck edges.

Considering the specified edges u and v in Figure 1. Edge u would not be regularly selected by ants during execution of Ant System, because there are plenty of alternative choices. In contrast, edge v will be regularly selected in different paths. That is because v is a bottleneck edge.



(a) edge u



(b) edge v

Figure 2: changes of pheromone for two edges of Figure 1.

Changes in pheromone values during time are shown for these two edges in Figure 2. To separate bottleneck edges from other edges on the shortest path, a criterion is needed. We call our proposed criterion *Roughness*. For a time series F, roughness is defined as follows:

$$R_F = \frac{\sigma_M^2}{(\max_i F_i - \min_i F_i)^2}, \tag{6}$$

where σ_M^2 is the variance of slope of pheromone diagram.

Using the definition of Roughness, each edge is given a score for being bottleneck. If edges on the shortest path are sorted due to roughness values in a list, then bottleneck edges have lowest amount of roughness. Next step is to separate bottleneck edges from others in the list. In the proposed method, it is suggested to analyze the growth of roughness in the sorted list of edges on the shortest path. The place where the value of diagram exceeds a specified proportion of lowest value, or the slope increments meaningfully, border of bottleneck and non-bottleneck edges are found.

Formally b is the index in the sorted array that separates bottleneck edges from other edges iff:

$$(Fail(b) = true) \wedge (\forall i < b : Fail(i) = false), \tag{7}$$

where $Fail(i)$ is a boolean function that tests if edge in index j is not a bottleneck and is defined as:

$$Fail(i) = (d_i > \tau_d.d_{init}) \vee (v_i > \tau_v.v_0), \tag{8}$$

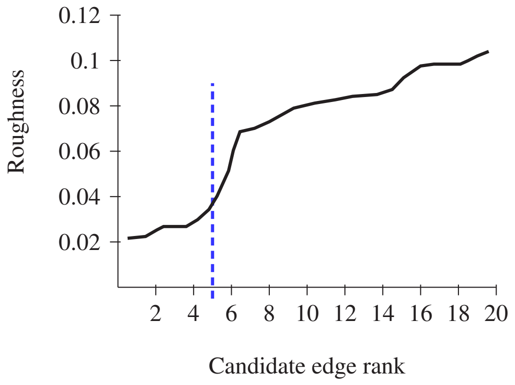


Figure 3: Roughness for sorted edges on the shortest path.

where v_i is the roughness for edge indexed i, d_i equals $v_i - v_{i-1}$ and d_{init} is the first non-zero d_i. Having bottleneck edges, it is easy to get the related vertices as sub-goals. Finally acquiring subsequent bottleneck edges does not help; thus for such cases all edges except for the edge with lowest value of roughness are removed. Algorithm 1 is a summary of the whole method.

Algorithm 1 The proposed method for sub-goal detection

```
1: Input: $(n_k, t_d, \alpha, \rho, \tau_d, \tau_v)$
2: Output: SubGoals: a list of sub-goals
3: Run Ant System $(\tau_d, \alpha, \rho)$ and have $SP$ with shortest path.
4: Sort $SP$ increasingly according to field $R_P$.
5: $v_0 \leftarrow SP[0].R_P$
6: for $i \leftarrow 1$ to length($SP$) do
7:     $d_{init} \leftarrow SP[i].R_P - SP[i-1].R_P$
8:     if $d_{init} \neq 0$ then
9:         exit the loop.
10:    end if
11: end for
12: for $b \leftarrow 1$ to length($SP$) do
13:    if $(SP[b].R_P - SP[b-1].R_P > \tau_d.d_{init}) \vee (SP[b].R_P > v_0.\tau_v)$ then
14:        exit the loop.
15:    end if
16: end for
17: for Adjacent :adjacent set of edges in $SP[0 \dots b-1]$ do
18:    $best \leftarrow argmin_i\{Adjacent.Edges[i].R_P\}$
19:    SubGoals.add(Adjacent.Edges[best].head)
20: end for
```

Skill Acquisition

After discovering useful sub-goals, next step is to create skills. First shortest path is broken into fragments which lie between selected bottleneck edges. Then communities which hold fragments of shortest path are found and skills for reaching the sub-goal in each community are formed. To generate partial policies for skills, experience replay method is applied and the created options are finally used via option framework.

Experimental Results

To evaluate performance of the proposed method, it has been tested in two standard environments: Taxi Driver and Playroom. Results of experiments are compared with those of a Q-Learning, betweenness and reformed label propagation (RLP) methods. According to betweenness method, those states which have a pivotal role in efficiently navigating the interaction graph, are useful subgoals (Şimşek 2008). Clearly speaking, this method defines subgoals as states that correspond to local maxima of betweenness on the interaction graph. RLP method (Davoodabadi and Beigy 2011) employs label propagation, which is a linear time algorithm, to find communities of the transition graph. Then uses modularity, a quality measure of a particular division of a graph (Newman and Girvan 2004), to merge smaller communities. Border states of the final communities are desirable subgoals.

Our experiments have two phases. In the *initial learning phase*, the agent explores the environment in a few episodes, and saves history of interactions in the transition graph. Then, it learns skills based on information of the transition graph. The length of the learning phase is chosen 7 episodes for taxi environment and 15 episodes for playroom. Next, in the *learning with options phase*, agent utilizes options learned in the previous phase and continues to interact with the environment in many episodes. The criterion for comparison is the number of steps taken during subsequent episodes, averaged on 40 runs. Q-learning agent uses ϵ-greedy method, with the following parameters: $\alpha = 0.05, \gamma = 0.9, \epsilon = 0.2$.

Taxi Environment

Taxi environment, is a 5×5 grid world containing a taxi and a passenger, as shown in Figure 4. Four special locations are specified as B, G, R and Y. Passenger's initial place and his destination are uniformly selected at random among these 4 locations. Taxi's task is to reach the passenger's initial place, to pick him up and to take him to his destination and to put him down. At each grid location, the taxi has six primitive actions: north, east, south, west, pick-up, and put-down. Each navigation action succeeds with probability 0.8, otherwise fails moving to the right or left side of intended direction. Action pick-up effects if taxi is in passenger's initial location and similarly action put-down is effective when passenger is in taxi and taxi is in destination location. Picking the passenger up in his initial location and putting him down in his destination will cause rewards +10 and +20 respectively. Other actions cost -1.

Figure 5 shows a diagram, depicting number of steps taken to reach goal state in consecutive episodes averaged over 40 runs. Parameters for the proposed method were set to: $n_t = 10, n_k = 25, \alpha = 0.9, \rho = 0.98, \tau_v = 1.01, \tau_d = 1.5$. The proposed method generates two skills in the taxi environment. One brings the taxi to the passenger's initial state and the other leads taxi to reach destination location. These two rational skills help the agent converge to optimal policy, much faster than the Q-Learning method. The reason of this improvement is that, when the taxi picks up the

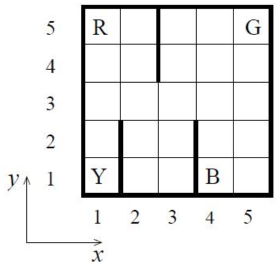


Figure 4: Taxi Driver environment

passenger, it can't put him down, unless it reaches the destination. Therefore, as the taxi finds the passenger more easily, it can complete its task efficiently. So the agent, which uses skills created by the proposed method, can find and deliver the passenger in fewer steps. As shown in figure 5, proposed method, acquires much better results as compared to betweenness and RLP methods. That's because of better skill formation of the proposed method. As it is mentioned in the experimental results of betweenness method (Şimşek 2008), in addition to real subgoals, some additional artificial subgoals are considered which cause a negative effect on results of this method. Simsek calls these additional subgoals as navigational bottleneck. RLP method works better than Q-learning and betweenness in earlier episodes after learning options. Totally, the proposed method stands significantly better than other methods.

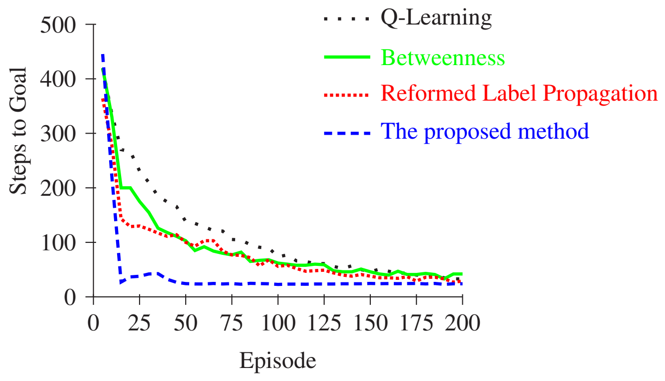


Figure 5: Comparison of Ant based, Reformed Label Propagation, betweenness and Q-learning in Taxi Driver environment.

Playroom Environment

In playroom environment, an agent interacts with a number of objects. A Markov version of this environment was introduced in (Şimşek 2008). Objects in the environment include a light switch, a ball, a button for turning music on and off, and a toy monkey. The agent has an eye, a hand and a marker that can be placed on objects. The agent's action set contains 1) look at a random object, 2) look at object at hand, 3) hold object it is looking at, 4) look at object marker is placed on, 5) place marker on object it is looking at, 6) move object in hand to location it is looking at, 7) turn over light switch, 8) press music button, 9) hit ball toward the marker. Two actions look at random object and look at object at hand succeed certainly but the other actions succeed with probability 0.75 and have no effect with probability 0.25. There are several rules in the environment. In order to work with an object, the agent must look at that object and hold it on his hand. When the agent turns the music on, it would become on if the light is on. If ball hits the bell, it rings for one time step and stops when the music is turned off. The toy monkey starts to make frightened sounds, if the bell is rung while the music is playing. The agent receives -1 reward for each action and +1000 for reaching the goal. In this experiment, 40 tasks were defined. Each of which was run in 200 episodes. In every task, the initial state was defined such that light, music and monkey scream are off and the agent's sensors (eye, hand and marker) detect nothing.

In order to have a fair comparison, we let each of algorithms to have an initial learning phase of 15 episodes. After that, agents create some options according to their history of interactions. In Figure 6 results of running the proposed method and the betweenness versus Q-learning are shown. Parameters for the proposed method were set to: $n_t = 200, n_k = 10, \alpha = 0.9, \rho = 0.98, \tau_v = 2.0, \tau_d = 1.5$. As concluded from Figures 6, convergence to the optimal policy occurs much faster in the proposed method in comparison with the conventional Q-Learning. All methods act nearly identical in the initial learning phase; however, after that, it can be seen that the ant based agent, can reach the goal with fewer steps.

In episode 80 the ant based agent, converges to approximate optimum policy, but the Q-learning agent converges after 200 episodes. These comparisons show that learned skills significantly improved Q-learning performance. Comparing results for the proposed method and betweenness method, implies no major dominance of each one over the other. The criterion, which is often used to compare performance of skill acquisition methods, is the convergence speed to the optimal policy, which seems a little better in the proposed method.

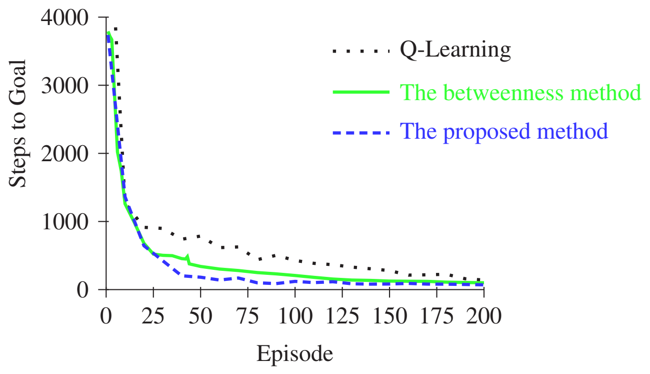


Figure 6: Comparison of Ant based skill learning, betweenness method and Q-learning in Play Room environment.

Discussion

Running time of the algorithm can be discussed as follows. Proposed algorithm for subgoal detection constructed from 2 main phases; running of ant system algorithm and finding the best candidates for bottleneck edges. In the first phase, each of n_k ants generate a path n_t times. Generation of each path is done by a pheromone directed DFS between start and goal nodes, which takes $\theta(n_t n_k(n+m))$ totally, where n and m denote number of nodes and edges in the transition graph respectively. This running time can be considered as $O(n+m)$ since n_t and n_k are small constants. To find the best candidates, we sort the edges of the shortest path that consists of $O(n)$ nodes. This procedure takes $\theta(n \log n)$ time. Then a linear search is taken place for finding the separation point of bottleneck edges and non-bottleneck edges. Considering the above discussion, the total running time would be $\theta(n \log n + m + n)$ which is equal to $\theta(n \log n + m)$. Comparing the time complexity of the proposed method with that of betweenness, which is $O(n^2 \log n + mn)$ (Şimşek 2008), concludes that the proposed method is a scalable way of skill acquisition and fits for lifelong learning, where scalability matters.

Conclusion

In this paper, a new graph theoretic based method for solving skill acquisition problem in reinforcement learning was proposed. The main idea of algorithm is to utilize Ant System algorithm to find shortest path between start and goal states in the transition graph. Using variance of slope of pheromone diagram of edges, a criterion was proposed. Edges that have lowest amount of roughness are bottleneck edges of the state space. The main advantage of the proposed method is that identified subgoals are placed on the optimal path between start and goal states, and thus all the discovered skills are necessary for the agent. Thus the proposed method avoids from useless skills, which cause the addition cost for the agent. Experimental results showed that our algorithm significantly improves the learning performance of the agent.

References

Davoodabadi, M., and Beigy, H. 2011. A new method for discovering subgoals and constructing options in reinforcement learning. In *proceeding of the 5th Indian International Conference on Artificial Intelligence (IICAI-11)*, 441–450.

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

Digney, B. 1998. Learning hierarchical control structures for multiple tasks and changing environments. In *proceedings of the 5th international conference on simulation of adaptive behavior on From animals to animats*, 321–330.

Dorigo, M.; Birattari, M.; and Stutzle, T. 2006. Ant colony optimization. *Computational Intelligence Magazine, IEEE* 1(4):28–39.

Ghafoorian, M. 2012. Automatic skill learning using community detection approach. Master's thesis, Sharif University of Technology.

Kazemitabar, S. J., and Beigy, H. 2008. Automatic discovery of subgoals in reinforcement learning using strongly connected components. In *proceeding of ICONIP (1)*, 829–834.

Kazemitabar, S. J., and Beigy, H. 2009. Using strongly connected components as a basis for autonomous skill acquisition in reinforcement learning. In *proceeding of Advances in Neural Networks–ISNN 2009*, 794–803.

Kheradmandian, G., and Rahmati, M. 2009. Automatic abstraction in reinforcement learning using data mining techniques. *Robotics and Autonomous Systems* 57(11):1119–1128.

Mannor, S.; Menache, I.; Hoze, A.; and Klein, U. 2004. Dynamic abstraction in reinforcement learning via clustering. In *proceedings of the 21st international conference on Machine learning*, 71–78.

McGovern, E. 2002. *Autonomous discovery of temporal abstractions from interaction with an environment*. Ph.D. Dissertation, University of Massachusetts Amherst.

Menache, I.; Mannor, S.; and Shimkin, N. 2002. Q-cut dynamic discovery of sub-goals in reinforcement learning. In *proceeding of 13th European Conference on Machine Learning: ECML 2002*, 187–195.

Newman, M., and Girvan, M. 2004. Finding and evaluating community structure in networks. *Physical review E* 69(2):026113.

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, 1043–1049.

Şimşek, Ö. 2008. *Behavioral building blocks for autonomous agents: description, identification, and learning*. Ph.D. Dissertation, University of Massachusetts Amherst.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.

Taghizadeh, N. 2011. Autonomous skill acquisition in reinforcement learning based on graph clustering. Master's thesis, Sharif University of Technology.