Autonomous Subgoal Discovery in Reinforcement Learning Agents using Bridgeness Centrality Measure

Mohammad Ali Imanian* and Parham Moradi**

* Department of Electrical Engineering, Harand Branch, Islamic Azad University, Isfahan, Iran. ma.imanian@gmain.com

** Assistant Professor, Computer Engineering and Information Technology Department, Engineering Faculty, University of Kurdistan, Sanandaj, Iran. p.moradi@uok.ac.ir

Abstract—The standard reinforcement learning methods do not scale up well for complex tasks and a solution to this drawback is to decompose a learning task into a set of subgoals, develop some skills based on the identified subgoals and then utilize the learnt skills. In this paper we propose a new approach for automatically identifying and evaluating subgoals based on bridgeness centrality measure. Investigating three benchmark problems, it is shown that how the proposed steps can significantly improve the learning performance of the agent.

Index Terms—Hierarchical Reinforcement Learning, Option, Skill Acquisition, Subgoal Discovery, Option, Graph Centrality Measure, Autonomous agents

I. INTRODUCTION

Reinforcement learning (RL) is an approach for autonomous agents to improve their performance through interactions with a dynamic environment. In particular, on each step of interaction, the agent performs an action that changes the state of the environment and the value of this transition is communicated to the agent through a scalar reinforcement signal, called reward. The agent’s goal is to devise a policy, i.e. assign probabilistic priority to each action on each state, which maximizes the expected discounted reward[1].

A key problem in the area of RL algorithms is that they are not solvable in complex tasks in a reasonable amount of time. Complex tasks are usually characterized by either a very large state space, or a lack of immediate reinforcement signals. There are two approaches to deal with these problems: the first approach is to apply low order approximation of value function[2] and the second approach is to divide a learning task into a set of simpler subtasks which leads to skills[3-6].

Many large tasks have some structure that allows them to be broken down into subtasks and represented more compactly. The smaller subtasks, are often solved more easily. The solutions to the subtasks may be combined to provide the solution for the original larger task. This decomposition decreases the size of an agent’s state space and consequently expedites the learning process.

Hence, instead of learning using individual primitive actions, an agent could potentially learn much faster if it could abstract the innumerable primitive actions to form high level behavioral macro-actions. Recent methods in RL allow an agent to plan, act, and learn with macro-actions [3-5]. A macro-action is a suitable sequence of primary actions which is known as skills. Utilizing skills, the performance of the agent is significantly improved while the learning phase is shortened dramatically.

The immediately raises the question of how the autonomous agent can develop skills automatically? There are various approaches to construct hierarchies of skills automatically [7-18]. A common approach is to define subtasks in the state space context. The learning agent identifies important states (subgoals), which are believed to possess some “strategic” importance and are worthwhile reaching. The agent learns sub-policies for reaching those key states and forms new skills[12-19].

There are various approaches to identify subgoals automatically. One approach is identifying those states which have been visited by the agent frequently or has gained high reward. One problem with frequency based approaches is that the agent needs excessive exploration of the environment in order to distinguish subgoals(bottlenecks) [17-19]. Some researchers analyze the learned policy for certain structural properties after the agent learns tasks[20-21]. Other researchers use graph theoretic approach to identify subgoals. In this approach the agent’s transition history is mapped to a graph and then, the states between strongly connected regions are identified as subgoals [12-16]. Menache et al[15]. have used the Max Flow-Min Cut algorithm to find bottleneck states while in [12, 14, 16] the state space is partitioned using some graph clustering algorithms. On the other hand, in [13] it is shown that the node centrality-based measures can be utilized as an effective measure not only for finding the subgoals but also for evaluating them. There are several centrality measures that numerically quantify the importance of nodes in a graph including betweenness centrality, closeness centrality, stress centrality and graph centrality [22]. Among these measures, the
node betweenness centrality measure has attracted much attention recently [13].

In comparison with other graph theoretic approaches, the node centrality based measures can be utilized as an effective measure not only for identifying the subgoals also for evaluating them [13].

Betweenness centrality measure considers only the global information of graph to score nodes. One can utilize both global and local information to identify subgoal states. In this paper we utilize “bridgeness” centrality measure to identify potential subgoals. This measure considers local and global information to score subgoal states [23]. We will show through simulation of three benchmark tasks, namely, “four-room grid”, “taxi driver” and “soccer simulation” that a bridgeness centrality measure performs better than the procedure based on closeness centrality and node betweenness centrality. Also in this paper we propose an algorithm to identify subgoal states based on graph centrality measure. The experiments show that utilizing this algorithm, subgoal states will be identified more precisely and the agent’s learning performance will be accelerated without any additional computational cost.

The rest of the paper is organized as follows: In section 2 we describe reinforcement learning basics and its extension to use option. The graph centrality measures are described in section 3. In section 4 the proposed method is described. The benchmark tasks, simulation and results are described in section 5, and section 6 contains the final discussion and the concluding remarks.

II. REINFORCEMENT LEARNING WITH OPTION

In this section, we define the basics of the RL with an extension to use options; see Sutton [3] for further details. The interaction of the agent and the environment can be represented using Markov Decision Process (MDP) framework. The MDP framework is a standard framework for learning and planning under uncertainty. A finite MDP is a tuple \( < S, A, T, R > \), where \( S \) is a finite set of states, \( A \) is a finite set of actions, \( T: S \times A \times S \rightarrow [0,1] \) is a state transition probability function and \( R: S \times A \rightarrow R \) is a reward function. At each decision stage, the agent observes a state \( s \in S \) and executes an action \( a \in A \) with probability \( T(s,a) \) which results in a state transition to \( s' \in S \). The agent obtains a scalar reward \( r \in R \) which is a (possibly stochastic) function of the current state and the action performed by the agent. The agent’s goal is to find a map from states to actions, called policy, which maximizes the expected discounted reward over time, \( E(\sum_{t=0}^{\infty} \gamma^t r_t) \), where \( \gamma < 1 \) is a discount factor and \( r_t \) is the reward obtained at time \( t \). To represent skills, we use the options framework [3]. Options are a generalization of primitive actions including temporally extended courses of actions. Even if adding options to the primitive actions set, expands the MDP representation, options offer a decomposition of the original task into subtasks, leading to a simplification of the global problem. A (Markov) option is a temporally-extended action, specified by a tuple \(< I, \pi, \beta >\) where \( I \) denotes the option’s initiation set, i.e., the set of states in which the option can be invoked; \( \pi \) is the option’s policy, mapping states belonging to \( I \) to a sequence of actions; \( \beta \) denotes the option’s termination condition, which \( \beta(s) \) denotes the probability that the option terminates in state \( s \).

In this paper, Macro-Q-Learning algorithm [3] (the extension of Q-Learning[24] algorithm with options) is used to optimize policies for both primitive actions and options. The Q-function that maps every state-action pair to the expected reward for taking this action at that state, is updated as follow:

\[
Q(s_t, o_t) := (1 - \alpha) Q(s_t, o_t) + \alpha \left( \gamma^t \max_{o_{t+1}} Q(s_{t+1}, o_{t+1}) + \sum_{k=0}^{t-1} r_{t+k} \gamma^k \right)
\]

Where \( \gamma \) is the actual duration of the option \( o_t \), \( \alpha \) is the learning rate. The update rule for a primitive action is similar with \( \gamma = 1 \). Performing a skill, the agent follows the policy of the skill until it reaches a termination node. Since the skills have longer execution time, the penalty assigned to a skill is higher than primitive actions and consequently developing useless skills leads to very low performance.

III. GRAPH CENTRALITY MEASURES

One can find different centrality measures in the literature including: betweenness, closeness, degree, eigenvector centrality and information centrality. In this section, we will briefly describe the definition of the closeness, betweenness and bridgeness centralities that are more applicable for RL problems.

A. Closeness Centrality Measure

Closeness centrality of a node \( u \), denoted by \( CC(u) \), is defined as the mean geodesic distance (i.e., the shortest path) between a node \( u \) and all other nodes reachable from it and it is defined as [22]:

\[
CC(u) = \frac{\sum_{v \in V} d(u,v)}{n - 1}
\]

Where \( n \) is the size of the graph and \( d(u,v) \) presents the length of the shortest path connecting \( u \) and \( v \) all other nodes reachable from it and it is defined as [22]:

\[
BC(u) = \frac{\sum_{u \neq t} \sigma_{st}(u)}{\sigma_{st}}
\]

Where \( \sigma_{st} \) is the total number of shortest paths between \( s \) and \( t \) and \( \sigma_{st}(u) \) is the number of shortest paths connecting \( s \) and \( t \) passing through \( u \).

B. Betweenness Centrality Measure

It is defined as the frequency that a node laid on a shortest path connecting two distinct nodes [25] i.e.,

\[
BC(u) = \frac{\sum_{u \neq t} \sigma_{st}(u)}{\sigma_{st}}
\]

C. Bridgeness Centrality Measure

A bridging node is a node lying between modules, i.e., a node connecting densely connected components in a graph. The bridging nodes in a graph are identified on the basis of their high value of bridgeness centrality relative to other nodes on the same graph. The bridgeness centrality of a node is the product of the betweenness centrality and the bridging coefficient, which measures the global and local features of a
node, respectively [23]. Specifically, the bridging centrality $SS(u)$ for node $v$ is defined by:

$$SS(u) = SB(u) \times BC(u) \quad (4)$$

Where $SB(u)$ denotes the betweenness centrality of node $u$ and $BC(u)$ presents the bridging coefficient of node $v$ that is defined as follows:

$$SB(u) = \frac{1}{d(u)} \sum_{i \in N(u)} \frac{1}{d(i)} \quad (5)$$

Where $d(u)$ is the degree of node $u$ and $N(u)$ denotes the neighbors of the node $u$. Betweenness centrality only considers global information of graph but the bridgeness centrality measure considers both global and local information of a node in the graph. Critical bottleneck nodes in the graph typically represent the limitation in connection to other nodes. This means that in the bottleneck nodes, the neighbors of the node also get high betweenness score and the bridging coefficient for this node will be more than its neighbors. It is worth mentioning that the proposed measure can be calculated with the same computational complexity of betweenness centrality using “Brandes” algorithm [22, 26] with minor modifications to involve bridging coefficient.

IV. PROPOSED METHOD

In this section our proposed method for automatic skill acquisition in reinforcement learning by autonomous agents will be described. The outline of the learning procedure is described in figure 1. First of all an environment is explored by “explore agents” and then a “head agent” translate the agents’ transition histories to a graph. After the graph constructed, our proposed algorithm is run on the graph to identify potential subgoal states. To overcome this problem, we proposed an algorithm that finds real bottleneck states as subgoal. The outline of proposed subgoal discovery is shown in the figure 2. In the first step the graph nodes are scored based on bridgeness centrality measure and the nodes with maximum bridgeness centrality measure is selected as subgoal. The neighbors of this node are also scored with high value and if high scored nodes are selected as subgoals, the bottleneck neighbors are also identified as subgoal states. To overcome this problem, the node with maximum score value is removed from the graph and the nodes of the new graph are scored again. This process is repeated until the graph is disconnected. At the end all of the nodes with maximum bridgeness centrality score are selected as subgoal states.

Algorithm 1

repeat
(1). Interact with environment and learn using Macro-Q-Learning.
(2). Save state transition history.
(3). If stop conditions are met then
(3.1). Translate the state transition history to a graph representation.
(3.2). Identify subgoals based on proposed subgoal discovery algorithm.
(3.3). Learn options to reach subgoals.
(3.4). Add new options to agents action space.
until no new states was found by agent

Figure 1: Outline of the Q-Learning with options, based on clustering.

A. Represent Agents Transition’s History as a Graph

In this paper, a multi agent system is proposed to explore the environment and translating agent’s history to a graph. In this system there are two types of agents, Explore Agents and a Head Agent. Every Explore Agent is doing a random task in the environment and learns using Macro-Q-Learning. Every new state transition obtained by these agents is sent to the Head Agent. The Head Agent collects all the Explore Agents transition histories and represents them as a graph. Each new states visited by agents becomes a node in graph and each observed transition $s_i \rightarrow s_j$ $(s_i, s_j \in S)$ is translated to an arc $(i, j)$ in the graph. In this paper we suppose that the graph is undirected and all weights of edges are set to one. The agents explore environment until no new state is observed by Explore Agents.

B. Identifying Subgoal States

After the environment is explored and represented as a graph, nodes in the graph are scored based on the considered centrality measure. There are two ways to identifying subgoal states. The first is selecting the top most scored nodes (above a threshold or k-top nodes where k is a predefined fixed number) considered as subgoal candidates. The second is selecting those nodes that scored more than a predefined threshold, considered as subgoal candidates. The scores should be normalized before identifying candidate subgoal nodes as shows in follows such as follows:

$$C_{Subgoal} = \{v \in V | \frac{C(v)}{\max_{v \in V} C(v)} > \tau\} \quad (6)$$

Where $C_{Subgoal}$ denotes subgoal candidates, $C(v)$ denotes score of a node $v$ based on the considered centrality measure i.e. bridgeness centrality measure, and $\tau$ denotes threshold for selecting subgoal candidates. Setting a correct threshold value is important, because if the threshold is set close to one, some right subgoal states may not be identified and if the threshold value is set to a value near zero, a lot of states will be identified as subgoal states. Some of these states may not be correct and skills built based on these subgoal states, will lead to bad effect on agent learning performance.

If different thresholds are set, different set of subgoals will be selected. Because the neighboring states of bottleneck state are scored also high value. So if subgoals are selected only based on setting threshold value on node centrality scores, some redundant states maybe selected as subgoal. To overcome this problem, we proposed an algorithm that finds real bottleneck states as subgoal. The outline of proposed subgoal discovery is shown in the figure 2. In the first step the graph nodes are scored based on bridgeness centrality measure and the node with maximum bridgeness centrality measure is selected as subgoal. The neighbors of this node are also scored with high value and if high scored nodes are selected as subgoals, the bottleneck neighbors are also identified as subgoal states. To overcome this problem, the node with maximum score value is removed from the graph and the nodes of the new graph are scored again. This process is repeated until the graph is disconnected. At the end all of the nodes with maximum bridgeness centrality score are selected as subgoal states.
C. Skill Generation

After identifying subgoal states, the next step is building agent skills. A skill is developed according to each identified set of nodes based on option framework [3]. Then it is added to the set of the actions of the agents after optimization of its corresponding policy. According to definition of an option, for each option, initiation set, termination set and policy should be defined. For each identified subgoal state, an option is defined and initiations set of the subgoal is the set of nodes that can start a skill to reach the subgoal. For each state and each subgoal we should determine that the corresponding skill can be executed or not. For this purpose, we define a betweenness based measure that calculates the dependency of the state to the subgoal state. This dependency measure is defined as follows:

$$\delta_{st}(g) = \sum_{t \in \tau} \frac{\sigma_{st}(t)}{\sigma_{st}}$$  \hspace{1cm} (7)

Where $\delta_{st}(g)$ shows the importance of subgoal $g$ from state $s$ perspective and $\sigma_{st}(g)$ denotes the number of shortest paths that passes through subgoal $g$ and $\sigma_{st}$ represents the total number of shortest paths between state $s$ and state $t$ in the corresponding graph. Based on this definition the initiation set for each subgoal $g$ is defined those nodes that have dependency value more than a predefined threshold. i.e.

$$I(g) = \{ s \in S | \delta_{st}(g) > t \}$$  \hspace{1cm} (8)

Where $I(g)$ denotes the initiation set for the subgoal $g$ and $t$ represent the predefined threshold. After defining initiation set for each subgoal the behavior of the option is optimized based on Q-Learning algorithm. Then the option is added to agent action space. Our proposed method for automatic option discovery is shown in figure 3.

D. Complexity Analysis

The complexity of bridgeness centrality is equal to the complexity of betweenness centrality, because the computational complexity of bridgeness coefficient ($SB(u)$) is $O(n)$ and the betweenness centrality may be computed in $O(nm)$ time and $O(n + m)$ space on unweighted graphs, where $n$ and $m$ are the number nodes and edges in the corresponding graph of explored states respectively. On weighted graphs the space requirements remains same but the time requirement increases to $O(nm + n^2 \log n)$ [22]. Because of the MDP properties and limitation of number of actions, in the most environments the corresponding graph of explored states are sparse, so the time complexity of bridgeness centrality will reduce to $O(n^2)$ and $O(n^2 \log n)$ on unweighted and weighted graphs respectively and the space complexity reduces to $O(n)$ in both weighted and unweighted graphs. In the “Algorithm 2” that was shown in figure 2, the bridgeness centrality of each node will be recalculated $k$ times where $k$ is the number of potential bottlenecks in the graph until the graph is disconnected. So the complexity of algorithm 2 will be accelerated to $O(kmn)$ and $O(k(nm + n^2 \log n))$ for unweighted and weighted graphs respectively. And this complexity will be reduced to $O(kn^2)$ and $O(kn^2 \log n)$ on unweighted and weighted graphs respectively for sparse graphs and the space complexity remains $O(n)$ in both weighted and unweighted graphs. If the number of subgoals $k$ is constant the complexity of “Algorithm 2” will be the same as complexity of bridgeness centrality.

V. EXPERIMENTAL RESULTS

We present an empirical evaluation of proposed algorithm aimed at understanding whether proposed method is effective in identifying the subgoal states in an environment and whether the skills it generates are useful. We present results in three environments: a four-room gridworld introduced by Sutton [3], the taxi grid world introduced by Dietterich [4] and soccer simulation that is more complex than four-room gridworld and taxi grid world. Complementary details and results about these domains are described in corresponding subsections.

A. Four room grid world

The four room grid world is shown in figure 1.a consists of four rooms connected to each other through four doors. The
agent is located at a randomly selected start point and asked to find a randomly selected goal point. The agent has four primitive actions, namely to move up, down, left and right. In the corresponding state transition graph, the cells are represented as nodes which are connected to their four neighbors. Then, we randomly select 60 tasks (namely 60 pairs <start, goal> locations). Each task is performed 100 times (episodes). The agent receives a reward of 1000 at the goal state and a reward -1 for all other states. To set the exploitation and exploration trade-off, our agents select out of policy, i.e. random actions with probability of 0.1, i.e. the agent uses an \( \epsilon \)-greedy policy with \( \epsilon = 0.1 \). The learning rate \( \alpha \) and the discount factor \( \gamma \) are set to 0.1 and 0.9 respectively.

### B. Taxi Grid World

The taxi task has been a popular illustrative problem for RL algorithms since its introduction by Dietterich [4]. This domain that is shown in figure 1.b, consists of a 5x5 grid with 4 special cells (RGBY). A passenger should be picked up from a cell and then dropped off in a destination cell, where the pickup and drop off nodes are two randomly chosen cells from the set of RGYB nodes. The corresponding state transition graph consists of two identical grids (one for the case that the taxi is searching for the passenger and one for the case when the taxi has picked up a passenger and is looking for the drop off location) that are connected through the pairs of corresponding RGYB nodes (the nodes where a change of state is possible). In each episode, the location of the taxi is chosen randomly. The taxi must pick up the passenger and deliver him, using the primitive actions up, down, left, right, Pickup and Putdown. For each iteration, a sequence of 300 episodes was considered. The taxi receives a reward of +20 for successfully delivering the passenger, -10 for attempting to pick up or drop off the passenger at incorrect locations and -1 for other actions. The other parameters were set the same as in the four-room grid problem.

![Image](image.png)

**Figure 4:** (a) Four room grid world domain (b) Taxi driver domain

### C. Soccer Simulation

As can be seen in figure 2, it consists of a 6*10 grid environment, two goals, a ball and two agents. At each episode one agent tries to own the ball and move to the opponent’s goal and score. The other agent tries defending and owning the ball. Each agent has five primitive actions: MoveLeft, MoveRight, MoveUp, MoveDown and Hold. The hold action causes the agent to hold the ball and remain in its location.

To score a goal an agent must hold the ball and move to one of two states in front of the opponent’s goal and perform a MoveRight (MoveLeft) action if the opponent’s goal is in the right(left) side of the field. When an agent scores a goal, the opponent owns the ball and two players are placed at specified location in front of their gate. On the other hand when two agents choose actions which leads to the situation that the agents must be placed in the same location owning of the ball is determined by following rules, a) If the agent which does not hold the ball is going to enter the other agent’s location, then with the probability of 0.8 the owning of the ball does not change and the locations of the players remain unchanged. b) If the agent that holds the ball is going to enter the location of the other with no moving player, then owning of the ball is changed and the locations of the players remain unchanged.

The environment will be specified with three state variable, each agents location will be specified by a state variable and the third state variable specifies the owner of the ball (value 0 is for owning of the first agent and value 1 is for the owning of the opponent). In other words, the corresponding graph has totally 60*60*2 (7200) nodes. The agents receive -1 reward for each action, +10 for owning the ball, -10 for missing the ball and +20 for scoring a goal. The agents uses an \( \epsilon \)-greedy policy with \( \epsilon = 0.1 \). The learning rate \( \alpha \) and the discount factor \( \gamma \) are set to 0.1 and 0.9 respectively.

![Image](image.png)

**Figure 5:** Soccer simulation world

### D. Results

In the first step, the corresponding graph of states was scored based on three different mentioned centrality-based scoring methods. Figure 6 represents the corresponding graph of four room grid world. The nodes of the graph are scored using bridgeness centrality measure, the lighter color of nodes corresponds higher centrality score. It can be seen that the bottleneck nodes scored higher values in comparison with other nodes. Figure 7 reports the scores assigned for the four-room grid task based on closeness(CC), betweenness (BC) and bridgeness (SS) methods respectively. Because of symmetry only the scores of the first 65 nodes have been shown in the figure. It can be seen that for the four room grid task, the bridgeness centrality measure assigns high scores to the door points (e.g. Nodes labeled with 25, 51, 62 and 78) distinctly comparing to other two measures. All the four doors are detected using the proposed method for a threshold larger than 96% but the other methods are not able to find the doors alone and they also assign high scores to door neighbors (eg. Nodes labeled with 24,26, 43, 54, 70, 77 and 79) and either face false acceptance, e.g. for threshold of 95%, CC found 10 additional nodes, or false rejection, e.g. BC discards the main doors for the thresholds larger than 82%.
These results show that the bridgeness centrality measure (SS) assigns high scores to the bottleneck nodes comparing with the CC and BC centrality measures. Figures 8.a, 8.b and 8.c show another representation for the result of the closeness, betweenness and bridgeness centrality scoring of the corresponding graphs of the four-room grid world. The lighter color of cells corresponds to the higher centrality scores. It can be seen that bridgeness centrality measure assigns high scores to door way nodes comparing with two other centrality measures. This measure also assigns low value to door neighbors (e.g. Nodes labeled with 24, 26, 43, 54, 70, 77 and 79).

As it was expected, the nodes around main subgoals, e.g. neighbors of hallway doors in the four-room grid world, have also high centrality scores. The next step is the elimination of the redundant subgoals which can be done by setting the threshold or, as it is proposed, by removing high score nodes as described in Algorithm 2. By varying the threshold, different number of nodes are extracted as subgoals. Figure 9 compares the number of candidate subgoals that identified in four-room grid world by different methods for different threshold values. When the threshold is set to 0.15, there are 104, 26 and 14 candidate subgoals extracted while using closeness, betweenness and bridgeness respectively. When threshold is set to 0.3, the bridgeness identified real bottlenecks while betweenness and closeness identified 104 and 13 candidate subgoals respectively. It can be seen that for different threshold values, the proposed algorithm extract four doors as subgoals. If we consider the redundant subgoals for creating skills we may create some complexity and additional penalties to an agent while obtaining no benefits.

The experiments were repeated for taxi grid world and the same qualitative results are reported. Figure 11 reports the results for the taxi driver domain. It can be seen that for this domain the bridgeness and betweenness centrality measures assign high scores to passenger locations that labeled with R, G, B and Y compared with their neighbors. But CC assigns the same score to the passenger locations and their neighbors so it will be difficult to distinct passenger locations with their neighbors. It can be seen in figure 11(d) that the proposed

Figure 6: Corresponding graph of four room grid world. The nodes are scored based on bridgeness centrality, the lighter color of nodes corresponds higher centrality scores.

Figure 7: Normalized assigned scores to the nodes using closeness, betweenness and bridgeness approaches for four room grid world.

Figure 8: The result of score assignments to the four room grid world nodes using (a) closeness(CC) (b) betweenness(BC) (c) bridgeness(SS) and (d) proposed algorithm. The lighter color, the higher score assigned.

Figure 9: The number of identified subgoals by applying closeness(CC), betweenness(BC), bridgeness(SS) and proposed algorithm as the function of the threshold values for four-room grid. Further descriptions are given in the text.
algorithm extracts exact passenger locations and a location (3,3) that is physical bottleneck, as final subgoals. Figure 10 compares the number of candidate subgoals that identified in taxi grid world by different methods when the threshold value slides from zero, i.e. extract all nodes, to one, i.e. extract no node. It should be noted that our proposed algorithm has effectively reduced the sensitivity of appropriate threshold selection.

To show the effectiveness of the proposed subgoal discovery algorithm, we repeat the experiments for the case that the agent extracts the subgoals by setting a threshold on closeness, betweenness and bridgeness scores or by applying the proposed subgoal discovery algorithm. The lighter color, the higher score assigned.

The experiments were repeated for the soccer simulation grid world and the same qualitative results were reported. Figure 14 shows the number of goals obtained by the agent for the different mentioned skill acquisition approaches comparing the situation that the agent uses standard Q-learning without using any skill. In these experiments the agent was able to gain 200 goals after 730 time steps when the subgoals were extracted using the proposed algorithm, while using Bridgeness, Betweenness and Closeness when the threshold is set to 0.5 the agent gained the same goals after 848, 932 and 1332 time steps respectively and using “without skill” approach the agent gained the same goals after 1484 time steps.
VI. CONCLUSION

In this paper, a graph theoretic based skill acquisition algorithm was presented. In brief, the main contributions of the proposed method are to utilize complex network theory measures for improving the subgoal identification process. In particular, the bridgeness centrality was defined and applied for candidate subgoal ranking and extraction. Then, the algorithm is proposed that removes redundant subgoals and extracts exact subgoals. Applying proposed method on three benchmark problems, results in improving the results of the skill acquisition process, i.e. subgoal identification, initiation set assignment and policy learning.

Here we report that the proposed method is also able to create skills incrementally. To do so, some temporary skills will be built based on explored states and in the next episodes by exploring more states, some new skills can be identified and then redundant skills or weaker ones will be removed. Further investigation on this issue and utilization of the proposed approach in more challenging environments are under progress.

From a computational complexity point of view, the proposed method run time is \(O(nm)\), where \(n\) and \(m\) are the number of nodes and edges in the corresponding graph of explored states, respectively. This complexity will be reduced to \(O(n^2)\) for sparse graphs. This result is the same as [13] and comparable with the method proposed in [15] with \(O(n^3)\) complexity, where \(n\) is the number of states, and [12] with \(O(n^2)\) complexity, where \(n\) is the number of states observed in the last episode.

The proposed method has a few numbers of adjustable parameters. While other methods such as L-Cut [12] and Relative Novelty [19] include manually tuned parameters, the proposed method has only an adjustable parameters, namely the threshold value \(t\) for identifying subgoal initiation sets.

REFERENCES