Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
October 2014, 21(5): 94–104
www.sciencedirect.com/science/journal/10058885
The Journal of China Universities of Posts and Telecommunications
http://jcupt.xsw.bupt.cn

Autonomic discovery of subgoals in hierarchical reinforcement learning

XIAO Ding (✉), LI Yi-tong, SHI Chuan

School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract

Option is a promising method to discover the hierarchical structure in reinforcement learning (RL) for learning acceleration. The key to option discovery is about how an agent can find useful subgoals autonomically among the passing trails. By analyzing the agent's actions in the trails, useful heuristics can be found. Not only does the agent pass subgoals more frequently, but also its effective actions are restricted in subgoals. As a consequence, the subgoals can be deemed as the most matching action-restricted states in the paths. In the grid-world environment, the concept of the unique-direction value reflecting the action-restricted property was introduced to find the most matching action-restricted states. The unique-direction-value (UDV) approach is chosen to form options offline and online autonomically. Experiments show that the approach can find subgoals correctly. Thus the Q-learning with options found on both offline and online process can accelerate learning significantly.

Keywords hierarchical reinforcement learning, option, Q-learning, subgoal, UDV

1 Introduction

RL is a promising approach to building autonomous agents and improving their performance with experiences. Although many tasks can be learnt by adopting the Markov decision process (MDP) framework using RL techniques, a fundamental problem of the standard RL algorithm is that, in practice, tasks cannot be solved in reasonable time. The difficulty in solving such tasks is usually due to the size of the state space and the lack of immediate reinforcement signals. Two common approaches were proposed to address these problems. The first approach is to apply generalization techniques involving low order approximations of the value function [1–2]. The other popular approach is to utilize hierarchical or related structures through task decomposition. The main idea of hierarchical reinforcement learning (HRL) methods in Ref. [3] is to decompose the learning task into simpler subtasks, and learn each of them independently. As a result, the overall task is better understood and the learning is accelerated. A major challenge of the approach is how to autonomically define the required decomposition, as in many cases the decomposition is not straight forward and cannot be obtained beforehand.

To address the problem, options (also known as temporally extended actions, skills, abstraction, or macro-actions) are introduced as closed-loop policies for sequences of actions to enable HRL [3–6]. The primary motivation of options is to permit one to add temporally-extended activities to the repertoire of choices available to an RL agent. A suitable set of options can then help improve the agent's efficiency in learning to solve large-scale problems. A popular approach to developing appropriate options is to identify subgoals and learn options for these subgoals autonomically. A subgoal is often a state or a region of state space, whereby reaching such a state or region is assumed to be able to facilitate the achievement of the overall goal of the task. A subgoal is simply like a doorway in a robot navigation scenario. When the robot wants to move out of a room, it must find the doorway first, which acts as a subgoal. If the agent can discover these subgoals and learn policies to reach them, it can use these policies for more effective exploration as well as refining overall policies more quickly. These

Received date: 08-11-2013
Corresponding author: XIAO Ding, E-mail: dxiao@bupt.edu.cn
DOI: 10.1016/S1005-8885(14)60337-X

subgoal policies can then be used to facilitate learning in similar tasks.

The purpose of this article is to autonomically find the target states which can usefully serve as subgoals. Different kinds of heuristics strategies or learning algorithms were applied to automatically discover subgoals. The strategies used in subgoal discovery can be roughly classified into five categories:

1) Some approaches choose states based on a non-typical reinforcement. For example, Digney [7] considered the states with a high reward gradient as subgoals. However, this approach may be not applicable in domains where there is a delayed reinforcement (e.g., a maze with eight rooms and one goal)

2) Another popular approach states are chosen according to their frequency of appearance [8– 9]. It is believed that the states that are often visited in the past are likely to be part of the agent's optimal path, and thus the exploration time may be reduced by finding local policies to reach those states. This method was refined by Mcgovern et al. [8] who proposed that the subgoals are states visited frequently on successful trajectories but not on unsuccessful ones. There are two disadvantages existing in the frequency based approaches: much inefficient frequency information may exist in the trajectories and the agent may need excessive exploration of the environment in order to distinguish between 'important' and 'regular' states.

3) Some other researches attempted to analyze the learnt policy for certain structural properties after the agent learning tasks [10–11]. For example, subgoals in Goel et al.'s work were discovered by studying the dynamics along the predecessor count curve and can include states that are not an integral part of the initial policy.

4) Graph theoretic is also used to identify subgoals. In these approaches, the agent's transition history is mapped to a graph and then the states between strongly connected regions are identified as subgoals [5,12–15]. For example, Menache et al. [13] used the max-flow/min-cut algorithm to find bottleneck states while the state space is partitioned by some graph clustering algorithms in Refs. [5,12,14,16].

5) Some other strategies were used to discover subgoals recently. Planning under uncertainty with macro-actions (PUMA) [17] combined the initial macro-action and successively shorter macro-actions for PUMA. Konidaris et al. [18] presented an abstraction selection algorithm to select a suitable abstraction from a library of available abstractions.

This article presented new heuristic information for subgoals, and thereby proposed a new subgoal-based method to form options autonomically in RL. By analyzing actions of the agent in a grid-world environment, we have found useful heuristic information to identify subgoals. That is, the agent must pass through subgoals in a certain direction and thus its effective actions in subgoals are not as arbitrary as those in other states. As a consequence, the subgoals are not only frequently visited but also action-restricted. Therefore, they can be defined as the most matching action-restricted states in the paths. That is, the direction of actions in subgoals is either uniform or inverted. To find the most matching action-restricted states for grid-world tasks, we would like to propose the concept of unique-direction value to denote the action-restricted heuristic. The UDV of a state in a path intuitionally reflects whether the path swerves in the state. It has been found that the approach can effectively distinguish subgoals from other states, especially the states near the subgoals. Experiments have proved the accuracy of the proposed approach in finding subgoals and the efficacy of learning options autonomically. We have also applied the UDV approach to form options offline and online. The offline option learning is to form the options through learning the random tasks beforehand, and learn the global optimal policies with these options, while the online option learning is to form options during the learning procedure, and switch to the learning with options autonomically. Experiments in both options show that with the help of options, the Q-learning can accelerate learning greatly. From these experiments, we have also analyzed how the option size and the time of generating options may affect the performances of the Q-learning.

It should be noted that the definition of subgoals is different from the existing methods. Firstly, it extends the definition of subgoals proposed by Stolle et al. [9] and Mcgovern et al. [8]. Subgoals in this article are regarded as states that are visited frequently as well as action-restricted. Thus our method is more likely to discover the correct subgoals when compared with those frequency-based methods [8–9]. The method is similar to some of the other state-transition-graph methods [5,13], in that all of us attempt to find the characteristics of subgoals from trails where the agent has passed; however, we try to apply a different technique and heuristics. The method here made use of the action-restricted property of subgoals and

proposed UDV concept to find these states. Autonomic subgoal discovery is often usually considered as a machine learning problem although Mcgovern et al. [8] regard it as a multiple-instance learning problem and some researchers in Refs. [12,19] consider it as a clustering problem. In this paper, we regard it as a path-matching problem.

The following is organized this way. In Sect. 2, the RL setup is described and extended to the use of options. Sect. 3 presents the detail of the autonomic subgoal discovery approach with unique-direction value. Sect. 4 applies the approach to form options offline, and validates it through two grid-world tasks. Sect. 5 describes the online application, and analyzes the factors that affect Q-learning with options through experimentation. Finally, we draw the conclusion in Sect. 6.

2 RL with options

RL is a computational approach to automating goal-directed learning and decision making [20]. It encompasses a broad range of methods for determining optimal ways of behaving in complex, uncertain and stochastic environments. Most RL researches are based on the formalism of MDPs. Although RL is by no means restricted to MDPs, this discrete-time, countable (in fact, usually finite) state and action formalism provides the simplest framework to study basic algorithms and their properties. Here we will briefly describe the well-known framework.

A discrete time MDP with a finite set of states S and a finite set of actions A works as follows. At each time step t, $t=1,2,...,$ the learning agent is in the state $s_t \in S$. The agent can choose an action a_t from a set of available actions at state s_t, $A(s_t)$, to cause a state transition to $s_{t+1} \in S$. The agent observes a scalar reward r_t that is a function of the current state and the action is performed by the agent. The agent's goal is to find a map from states to actions, called a policy, which maximizes the expected discounted reward over time, $\mathrm{E}\left\{\sum_{t=0}^{\infty}\gamma^t r_t\right\}$, where $\gamma<1$ is the discount factor and expectation is taken with respect to the random policy of the agent. A common solution strategy is to approximate the optimal action-value function, or Q-function, which maps each state and action to the maximum expected return starting from the given state and action and thereafter always takes the best actions [6].

Recently, macro Q-learning [21] was proposed to fasten the learning process, which has extended Q-learning with options. An option is a sequence of actions that are executed by the agent until a termination condition is met. Formally, an option is defined by a triplet $\langle I,\pi,\beta\rangle$, where I is the option's input set, i.e., all the states by which the option can be initiated; π is the option's policy, mapping states belonging to I to a sequence of actions; β is the termination conditions over states, i.e., $\beta(s)$ enotes the termination probability of the option when reaching state s. The subgoal is usually regarded as the terminating state. As a consequence, the autonomic subgoal discovery is crucial to the formation of options. While the agent is following an option, it will not stop until the option terminates. When not following an option, the agent can either choose a primitive action or initiate an option. The updated rule for an option o, initiated at state s, is

$$Q(s,o)\leftarrow Q(s,o)+\alpha[r+\gamma^{\tau}\max_{o\in O}Q(s',o)-Q(s,o)]$$

where τ denotes the number of time steps elapsing between s and s', r denotes the cumulative discounted reward over this time, and it is implicit that the step-size parameter α is the learning rate that may depend arbitrarily on the states, option, and time steps. The estimated $Q(s,o)$ converges to the optimal value function over options, $Q_O^*(s,o)$ for all $s\in S$ and $o\in O$ under conditions similar to those for conventional Q-learning. The updated rule for a primitive action is similar with $\tau=1$.

3 Autonomic subgoal discovery using UDV

A direct option discovery approach is to generate new options and let the agent test them by adding them to its set of actions. However, due to the great size of the state space and the large number of optional actions and options, such approach will inevitably cause inefficiency. So it is necessary to find small quantity yet high quality subgoals. Observing the actions of an agent in subgoals, we can find some novel properties that may be useful to discover subgoals.

3.1 Action properties of subgoals

Take an example of scenario in which an agent goes out of a room to a restaurant for lunch. You may wander in the room, moving to whatever directions possible: left, right,

forward or backward. But once he reaches the doorway, he can only go either into or out of the room; otherwise you will bump onto the doorframe. Since the ultimate goal is the restaurant for lunch, the agent will definitely go out of the room at the doorway. Obviously, the doorway is a critical subgoal in the context. It must be passed through if the agent wants to go out of (or into) a room. Furthermore, the effective actions in the doorway are restricted to going into and out, not turning left or right.

A clearer explanation in the grid-world environment is shown in Fig. 1. In the environment, the black cell represents the blocking wall and the white cell represents the free space to explore. Suppose that the policies of the whole state space was learnt, we randomly select a set of start state and target state as the agent's tasks. The known policies allow us to find the shortest path from the start state to the target state, and then record the action of each state in the path. Fig. 1 illustrates the four random tasks and their shortest paths. It is clear that the two useful subgoals in the figure are $s(3,3)$ and $s(6,7)$. Here, $s(x,y)$ means the state in the grid-world in the xth row and yth column (the cell in top left corner is $s(0,0)$). Every path shown in the figure passes through one of these subgoals. Actions in the subgoals are either in the uniform direction (e.g., $s(3,3)$) or in the inverted direction (e.g., $s(6,7)$). However, the actions in other states are not restricted, such as $s(4,6)$ and $s(7,7)$.

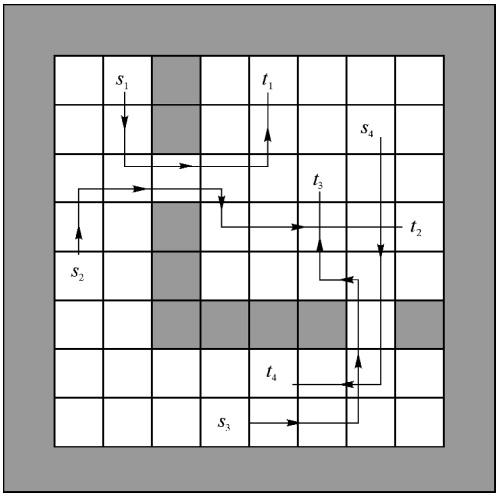


Fig. 1 Illustration of action properties of subgoals in 10×10 grid-world

Through analyzing the effective actions of agents in subgoals, we can conclude that the actions in subgoals have the following two properties.

Propery 1 The subgoals are the states through which trajectories connecting start and target states often pass.

This property implies that the subgoals are visited frequently. Many frequency-based approaches are based on this property. For example, in the work by Stolle et al. [9], if states occur frequently on trajectories that represent solutions to random tasks, these states may be important. The property is called the high-frequency property.

Propery 2 The subgoals are the states where the effective actions of an agent are usually restricted.

To achieve the task, the agent must pass through the subgoals in the direction from the start state to the target state. The effective actions in the subgoals are restricted unlike those in other states. The property is also called the action-restricted property. It should be noted again that the effective actions are emphasized here because the agent can explore in any possible directions in subgoals, and some noneffective directions may lead to bumping onto the wall. The action-restricted property is important as it can distinguish subgoals with other states through their actions. However, it is usually ignored by many contemporary methods.

According to Property 1, the paths passing through the same subgoal should match with the subgoal with a high frequency. However, it is not correct that high-frequency matching states are subgoals. Property 2 is an effective method to distinguish true subgoals from other high-frequency matching states. The effective actions of the agent in subgoals are restricted in uniform or inverted directions (no turns or swerves possible). Taking Fig. 1 for example, path $<s_1, t_1>$ and path $<s_2, t_2>$ match in states from $s(3,2)$ to $s(3,4)$, and path $<s_3, t_3>$ and path $<s_4, t_4>$ match in states from $s(5,7)$ to $s(7,7)$. Among all these six states, only $s(3,3)$ and $s(6,7)$ are action-restricted. Thus, only these two states are considered as the potential subgoals. Therefore, Property 2 is optimal in looking for the most matching action-restricted states in paths as subgoals.

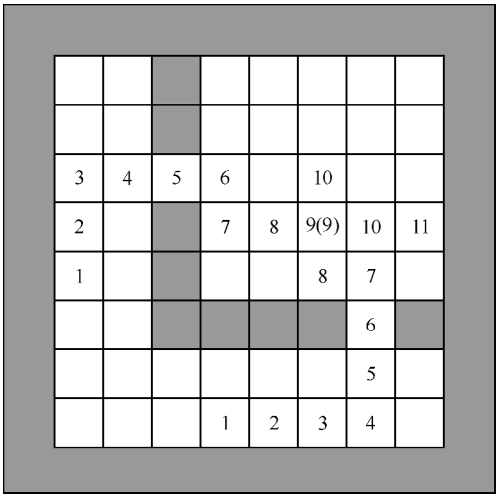
3.2 The UDV approach

In current environment, it is hard to find the most matching action-restricted states in paths due to the high computational complexity. However, it is possible to solve the problem in the grid-world environment since it is regular. Here, the authors want to propose a novel concept of UDV, which can effectively denote the information of action directions. Through the UDV, the subgoals can be

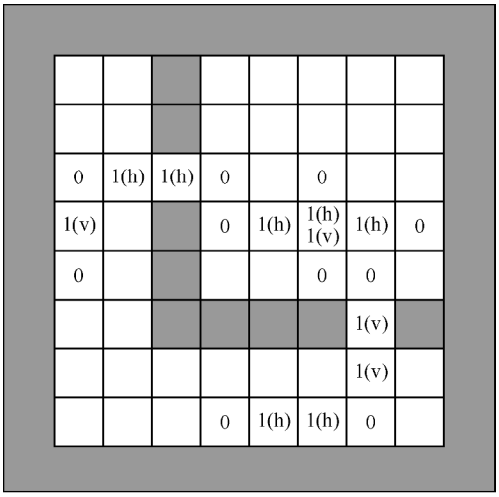
easily calculated as the states with max UDV in paths. To calculate the UDV, paths should be mapped into the grid-world. To generate a path, we can calculate the UDV of each state in the path by utilizing the trait of the grid-world. To find all paths, we can accumulate the UDV of each state and select the maximal ones as potential subgoals. The detailed approach is described as follows.

3.2.1 Mapping a path

To map a path into the grid-world, the states in the path can be labeled following their sequences. If a state is visited many times, it only records the last sequence. As Fig. 2(a) shows, it maps two paths <s_2, t_2> and <s_3, t_3> into the grid-world.



(a) Mapping path <s_2, t_2 >and <s_3, t_3> in Fig. 1 into the grid-world



h−horizontal UDV v−vertical UDV

(b) The UDV of states in Fig. 2(a)

Fig. 2 The detailed approach to find all paths

3.2.2 Calculating the UDV

As was pointed, the effective actions in subgoals are restricted to left/right or up/down in the grid-world, compared to left/right/up/down in other states. Here, we want to use the unique-direction value of a state to denote action directions of states. Each state in a path has a horizontal unique-direction value (HUDV) and a vertical unique-direction value (VUDV) that are denoted as $V_{\text{IsHorDir}}(s)$ and $V_{\text{IsVerDir}}(s)$ respectively, which can be defined as follows:

$$V_{\text{IsHorDir}}(s(x,y)) = \begin{cases} 0; & 2S_{\text{Seq}}(s(x,y)) \neq S_{\text{Seq}}(s(x,y-1)) + S_{\text{Seq}}(s(x,y+1)) \\ 1; & 2S_{\text{Seq}}(s(x,y)) = S_{\text{Seq}}(s(x,y-1)) + S_{\text{Seq}}(s(x,y+1)) \end{cases} \tag{1}$$

$$V_{\text{IsVerDir}}(s(x,y)) = \begin{cases} 0; & 2S_{\text{Seq}}(s(x,y)) \neq S_{\text{Seq}}(s(x+1,y)) + S_{\text{Seq}}(s(x-1,y)) \\ 1; & 2S_{\text{Seq}}(s(x,y)) = S_{\text{Seq}}(s(x+1,y)) + S_{\text{Seq}}(s(x-1,y)) \end{cases} \tag{2}$$

where $S_{\text{Seq}}(s(x,y))$ denotes the sequence value of $s(x,y)$ in a path. Take Fig. 2(a) as an example, $S_{\text{Seq}}(s(3,3)) = 5$ and $S_{\text{Seq}}(s(6,7)) = 6$.

According to the definition, $V_{\text{IsHorDir}}(s) = 1$ means the states on the left side and right side of s are the preceding and successive states of s in the path, that is, the path passes through s in the horizontal direction. Similarly, $V_{\text{IsVerDir}}(s) = 1$ means the states under s and above s are the preceding and successive states of s in the path, which is the path passing through s in the vertical direction.

Because the subgoals must be passed through, the UDV of subgoals in a path is always 1. Since the effective actions in the subgoal are restricted, either the HUDV or the VUDV is 1 in the subgoals. Consequently, the subgoals have the following property.

Property 3 If a state s is a subgoal, for all paths passing s, $V_{\text{IsHorDir}}(s) = 1$, $V_{\text{IsVerDir}}(s) = 0$, $V_{\text{IsHorDir}}(s) = 0$ and $V_{\text{IsVerDir}}(s) = 1$.

Compared with subgoals, the UDV of other states in the path is 1 or 0, and their HUDV and VUDV may both be 1. Fig. 2(b) shows the UDV of states in Fig. 2(a). As Fig. 2(b) shows, for the subgoals $s(3,3)$ and $s(6,7)$, $V_{\text{IsHorDir}}(s(3,3)) = 1$ and $V_{\text{IsVerDir}}(s(6,7)) = 1$. For the non-subgoal state $s(4,6)$, $V_{\text{IsHorDir}}(s(4,6)) = 1$ and $V_{\text{IsVerDir}}(s(4,6)) = 1$.

In fact, the UDV of a state reflects whether the path swerves in the state intuitionally. A state's UDV being 0 means the path swerves in the state. As the scenario has illustrated, one can wander in the room. Similarly, the path can swerve in a room, which means the UDV of the non-subgoal states in the path can be 0. Different paths can pass the same position in the room in different directions. That is, the HUDV and VUDV of the non-subgoal states can both be 1. Then the path cannot swerve in the subgoal, namely, the UDV of the subgoals in the path must be 1, or the agent will bump onto the wall. And the path must pass through a subgoal in a certain direction, that is, either the HUDV or the VUDV is 1 in the subgoal.

3.2.3 Accumulating UDV of all paths

For the mapping of all paths, we can calculate the accumulated UDV of each state. $V_{\mathrm{HorDir}}(s)$ and $V_{\mathrm{VerDir}}(s)$ represent the sum of the HUDV and VUDV in state s respectively.

$$V_{\mathrm{HorDir}}(s)=\sum_{l\in L}V_{\mathrm{IsHorDir}}(s_l) \tag{3}$$

$$V_{\mathrm{VerDir}}(s)=\sum_{l\in L}V_{\mathrm{IsVerDir}}(s_l) \tag{4}$$

where, L is the path set, l is one path, s_l means the state s in path l.

The accumulated UDV of a state s in all paths is denoted as $V_{\mathrm{UniDir}}(s)$, which can be defined as follows:

$$V_{\mathrm{UniDir}}(s)=|V_{\mathrm{HorDir}}(s)-V_{\mathrm{VerDir}}(s)| \tag{5}$$

Finally, we select the states with the maximum accumulated UDV as subgoal set.

$$S_{\mathrm{SG}}=\max_{N_{\mathrm{sg}}}V_{\mathrm{UniDir}}(s);\quad s\in S \tag{6}$$

where S_{SG} represents the subgoal set and S represents the state set. If there are N_{sg} subgoals, we select the first N_{sg} states with maximum accumulated UDV as subgoals.

$V_{\mathrm{UniDir}}(s)$ can distinguish the subgoals and other states for two reasons. On one hand, according to Property 1, subgoals usually have larger visiting frequency. That is, the UDV of subgoals is large. On the other hand, according to Property 2, the subgoals are different from other states in restricted actions, which makes the accumulated UDV of subgoals larger than high-frequency non-subgoals states.

For the $m\times n$ grid-world, there are $N_{\mathrm{s}}=m\times n$ states. Each path is so far mapped into the grid-world. There are L paths, so the storage space complexity is $O(LN_{\mathrm{s}})$. Each state also needs to be mapped into the grid-world, and its unique-direction value and the accumulated unique-direction value need to be caculated. And thus, for L paths and N_{s} states, the approach needs $3\times L\times N_{\mathrm{s}}$ basic operations. Consequently, the run-time complexity is $O(LN_{\mathrm{s}})$. The run-time and space complexities are both linear with the number of paths and states.

4 Application in forming options offline

4.1 Forming options offline

As illustrated above, the UDV approach can be used to autonomically discover the subgoals that are the crucial components of options. In this section, we will apply the approach to form options offline, which means that the agent explores the environment to learn options ahead of time. During this time, the option discovery process is based on a series of random tasks in this environment, and the agent learns to find options from random tasks. Then the options learnt can be used to accelerate these kinds of learning tasks.

In the grid-world environment, the whole state space can be divided into several subspaces. The states in these subspaces are closely connected, and obviously, these subspaces are divisional. Some in-between states connect these subspaces, which are then regarded as the subgoals. The definition of option requires us to find the input set, the internal policies in the option, and the termination condition. Suppose each subspace in the state space corresponds to an option, the process can be constructed as follows:

1) The subgoals in the subspaces can be regarded as the terminate state of the option.

2) The other states except the subgoals in the subspace can be regarded as the input set I.

3) The internal policies in the option can be learnt with Q-learning when the input set and terminate state are determinate.

The option discovery algorithm is described in Algorithm 1 and Algorithm 2 in detail.

Algorithm 1 The discovery of subgoals with UDV approach in a grid-world environment.

```
DisSubGoal(){
    Select a number of learning tasks at random;
    For each task
        Perform N_train episodes of Q-learning;
```

```
        Perform N_test episodes and record the paths;
        Map these paths into the grid-world;
        For each path and state s in it
            Calculate V_VerDir(s) and V_HorDir(s);
    For each state s
        Calculate V_UniDir(s) for all tasks;
    Select the N_sg subgoals with maximum V_UniDir(s);
}
```

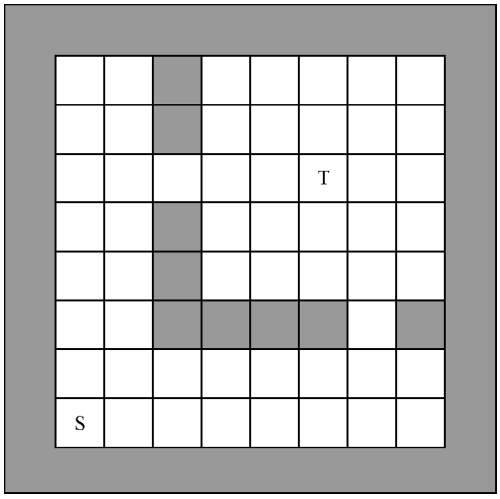
Once the options are found, we can use SMDP Q-learning, as described in Sect. 2, in order to learn the optimal policies over options. We may notice that the paths may not visit all initial states, so we cannot guarantee that all initial states in the subspace are in the input set. In order to learn from the arbitrary states for the agent, we should add all primitive actions as one-step options to the option set.

Algorithm 2 Automatic creation of option with UDV in a grid world environment.

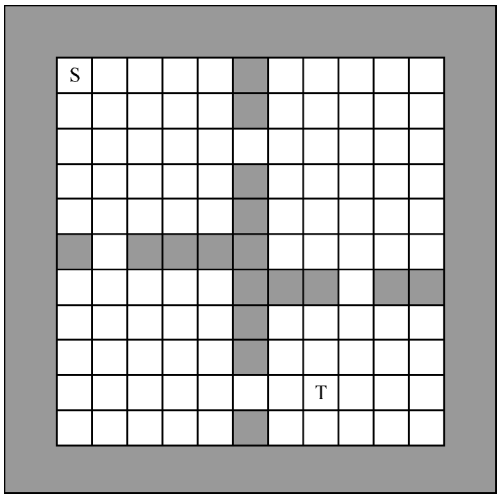
```
CreateOptionOffline(){
    Run DisSubGoal() to get N_sg subgoals SG;
    For each subgoal s in SG
        Define a new option o whose terminate state is the subgoal
          s;
        For each path passing through s
            Set the states between its preceding and successive
              subgoal as its input set;
        Set the input I of o with the combination of all input set of s;
        Learn the internal policies of o to reach the subgoal s;
}
```

4.2 Experimental study in offline application

In the experiment, two grid-world navigation tasks were used to test our proposed algorithm: a 10×10 grid-world with 2 subgoals, and a 13×13 grid-world with 4 subgoals as shown in Fig. 3. The subgoals are hallways in the grid-world (i.e. bottleneck states in the environment), since trajectories passing from one room to another have to pass through the hallways. The navigation task intends to find the shortest path from the start state (marked 'S' in the grid-world) to the target state (marked 'T'). In such context, the algorithm is expected to find the subgoals first, then options for going to the subgoals, and finally the shortest way of navigation over the options.



(a) Rooms environment in 10×10 grid-world



(b) Rooms environment in 13×13 grid-world

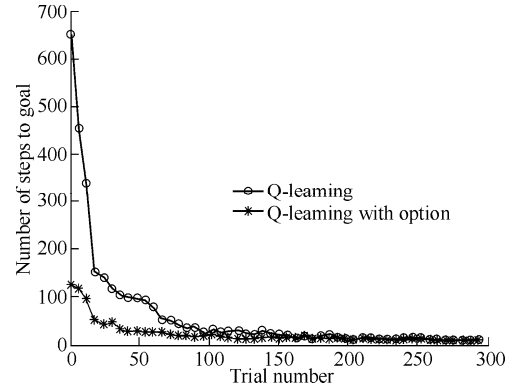
Fig. 3 Two grid-world navigation tasks used in our experiment

The state is in the current cell position. There are four deterministic primitive actions of the agent: up, down, left and right. If the agent attempts to move into a wall, it stays in the same position, no penalty is incurred. The discount factor $\gamma = 0.9$ and there are no intermediate rewards anywhere. The agent can only obtain a reward upon entering a designated goal state and the reward is 100.

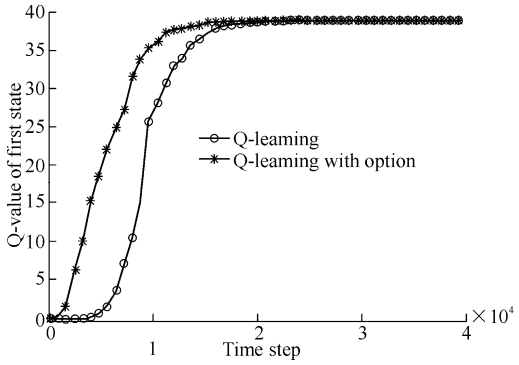
The algorithm runs independently for 20 times, and the results are the average values. During each run, the option discovery algorithm is used to find as many options as the subgoals. During the option discovery stage, the 25% of the states are used as potential start and target states for random tasks. We use $N_{\text{train}} = 100$ episodes to learn Q-values for each pair of start and target states. After learning, we use $N_{\text{test}} = 10$ episodes to generate the UDV. Once the options are learnt, we compare the performance

of a learning agent using primitive actions only with the performance of an agent using both primitive actions and options. Both agents use a learning rate $\alpha = 0.1$ and ε-greedy policy for generating behavior with $\varepsilon = 0.1$.

The authors did many experiments in different environments and observed that the UDV approach can find the subgoals correctly. Figs. 4(a) and 5(a) illustrate the expected number of steps to the goal for the two algorithms of Q-learning and Q-learning with options in the two navigation tasks. It can be found that Q-learning with options achieves optimality much earlier than standard Q-learning. The advantage of using options is further demonstrated in Figs. 4(b) and 5(b).



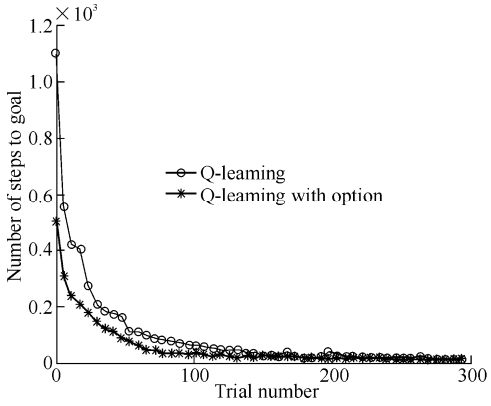
(a) Average number of steps to goal



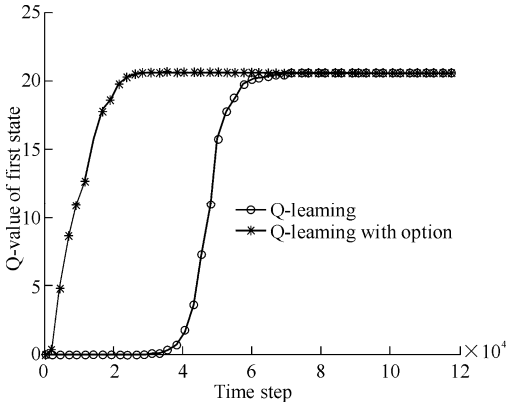
(b) The average Q-value of the initial state

Fig. 4 Experiment results in 10×10 grid-world

Here we present the Q-value of the start state as a function of time step, where each interaction with the environment takes one unit of time. The Q-value of the start state of Q-learning with options increases much faster than that of Q-learning. In these two grid environments and by two criteria, Q-learning with options has always achieved better performances. It should be noted that these two methods (i.e., Q-learning and Q-learning with options) have the same settings except for the option learning, so we think the better performances of Q-learning with options attribute to the option learning process. Generally, with the help of options, the agent can find the subgoals easily and reach the goal more quickly. That is the reason why the Q-value of the first state reaches the maximum value faster compared with Q-learning.



(a) Average number of steps to goal



(b) The Q-value of the initial state

Fig. 5 Experiment results in 13×13 grid-world.

5 Application in forming options online

In the offline application, we assume that the agent can explore the environment to form options beforehand. However, the assumption is unpractical in many real applications. Moreover, the agent usually spends much time excessively exploring the environment in order to discover valuable subgoals. As a consequence, the autonomic options discovery during the learning process is more important and useful. In this section, we will apply the unique-direction-value approach to form options online.

5.1 Forming options online

In the experiment, we added the autonomic option

generation procedure and combined it with macro-Q-learning. The outline of the learning procedure is described in Algorithm 3. The atomic action is considered as an option with one step. Compared with the primitive Q-learning, Algorithm 3 adds the process that generates options autonomically. The time of generating options faces a trade-off. The options discovery algorithm is performed early in the learning process, where the impact on the exploration would be the most significant. On the other hand, if options are performed too early, the information obtained may not suffice to generate meaningful subgoals, and the resulting options would contribute less to the learning effort. To describe the condition, we define two parameters in the algorithm. The lower boundary parameter is the beginning time that starts to record the paths; the up boundary parameter is the ending time that finishes recording the paths. They are both the percentages of the running generations. The following experiments examined their effect on the algorithm. The conditions for generating options should in general be problem-dependent and call for further study. The similar issue was studied in Mannor et al. [12].

Algorithm 3 Q-learning algorithm with option.

```
QLearnOption(){
    Learn the optimal policies with Q-learning;
    If condition is satisfied
        Call CreateOptionOnline();
        Add options into action state space;
    Learn the optimal policies with Q-learning based on options;
}
```

Then let us consider how to generate options autonomically. The autonomic option discovery algorithm online is described in Algorithm 4. The basic idea is that the agent calculates the UDV of each state in the paths to discover the subgoals, and then analyzes the paths to find the input set I, and finally forms the internal policies π. The parameter λ in Algorithm 4 is used to control the size of option. The appropriate option size is also a trade-off. Too large option may have negative effects on performances, whereas too small option may not accelerate the learning significantly. However, the appropriate λ is problem-dependent and needs further study. The authors analyzed how λ affects the performance in the following experiments.

Algorithm 4 Online create option with UDV in a grid world environment.

```
CreateOptionOnline(){
    For each path recorded
        Map the path into the grid-world;
        For each state s in the path
            Calculate V_IsVerDir(s) and V_IsHorDir(s);
    For each state s existing in paths
        Calculate V_VerDir(s), V_HorDir(s) and V_UniDir(s);
    Select N_sg states with maximum V_UniDir(s) as subgoals SG;
    For each subgoal s in SG
        Define a new option o whose terminate state is the subgoal
          s;
        For each path passing through s
            Set the states whose distance from s is smaller than λ
              as its input set;
        Set the input I of o with the combination of all input set of s;
        Learn the internal policies of o to reach the subgoal s;
}
```

5.2 Experimental study in online application

Fig. 6 illustrates a more complex task, where a grid-world navigation task with six subgoals is used to validate the proposed algorithms. In Fig. 6, 1-6 are the subgoals in the environment, and G is the goal state. The navigation task intends to find the optimal policies that the agent travels from the other states to the goal state G.

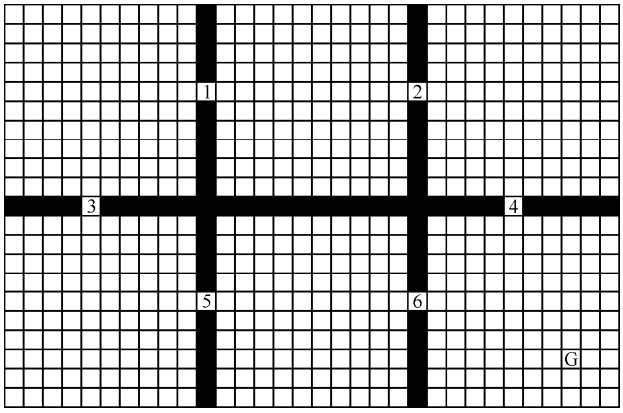
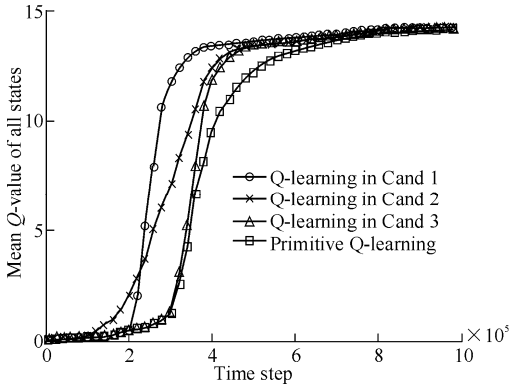


Fig. 6 Rooms environment in 21 × 32 grid world.

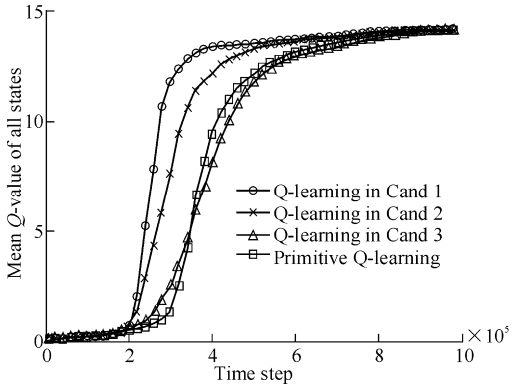
The experiment environment and the parameters are the same with that in Sect. 4.2. When the agent reaches the goal successfully, it will randomly select an initial state to continue to learn the optimal policies. In the experiments, the mean Q-value of all states is used as the function of the time step. The experiments were designed to compare the primitive Q-learning with options based Q-learning.

Fig. 7(a) illustrates the relation between the mean Q-value of all states and time step in different conditions that generate options. Fig. 7(b) illustrates the relation

between the mean Q-value of all states and time step in different sizes of options. λ is $R/3$, where R is the width of the grid-world (i.e., 32).



(a) Different time generating options



(b) Different size of options

Fig. 7 The relation between the mean Q-value of all states and time step in 21×32 grid world

From Fig. 7(a) we can observe that there are no obvious differences in the four algorithms before the options are generated, since they all are the primitive Q-learning. After the options are generated, the algorithms with options have better performance than those without options, since they reach the max Q-value faster. At the same time, we have also found that [lower boundary, up boundary] have a distinct effect on the performance. In the condition Cand 1, the parameter is [0.1, 0.2], the algorithm has the most significant performance improvement. Since some paths have been learnt in this condition, the valuable information contained in these recorded paths is useful to generate meaningful options. It is found that these good options can accelerate the Q-learning obviously. In the condition Cand 2, the parameter is [0, 0.1], the paths are recorded from the beginning. And thus they contain less valuable information, so the options generated by them may not be valuable. As a consequence, although these options can accelerate Q-learning, the result is not as remarkable as that in Cand 1. In the condition Cand 3, the parameter is [0.2, 0.3], it is a little late to generate options. Although options can still fasten the Q-learning to some extent, the improvement is limited, since the primitive Q-learning has begun to converge quickly. In all, the Q-learning with options is sensitive to the time of generating options. It should be pointed out that the primitive Q-learning has tardy process in the beginning phase, quick performance improvement in the middle phase, and slow convergence in the end. This shows that perhaps we should utilize the characteristic and select the appropriate time to generate valuable options. In our algorithm, the appropriate time is that it begins to record the paths after some paths are learnt and it generates the options before fast Q-learning convergence.

In the experiment, lower boundary is 0.1, and, up boundary is 0.2. Observing Fig. 7(b), we have found that the four algorithms have close performance before generating options, whereas the performance of the algorithms with options is better than those without options after the options are generated. In the condition Cand 1 (i.e., $\lambda = R/3$), the algorithm has the best performance. With the help of options, it converges to the optimal value quickly. This might be due to the fact that when $\lambda = R/3$, the initial states of an option are exactly the states in both sides of the subgoals. Such option has an appropriate size, so its performance is the best one. In the condition Cand 2 (i.e., $\lambda = R$), its performance is better than that of the primitive Q-learning (i.e., Q-learning without options) before 3.5×10^5 time step, whereas it is not the case after the time step. When $\lambda = R$, the size of options is too large, which may have a negative effect on the algorithm. In the condition Cand 3 (i.e., $\lambda = R/6$), the algorithm has also improved the performance obviously, whereas it is still worse than that in Cand 1. When $\lambda = R/6$, the size of options is a little small, so these options may not show their potential sufficiently. The experiments have shown that the proposed algorithm is also sensitive to the size of options. The selection of the appropriate option size is problem-dependent. In the grid-world environment, it is better for the option to incorporate the states in subspace (except the subgoal) into its initial states.

6 Conclusions

This article presented action-restricted heuristics for

subgoals and proposed a novel UDV approach to autonomically discover subgoals. Subgoals in this paper are regarded as the most matching action-restricted states in the paths. In grid-world tasks, the action-restricted heuristics can be roughly simplified to the UDV, which can be adopted to distinguish subgoals from other states. Thus, we propose the UDV approach for autonomic subgoal discovery and illustrated its application with two cases. In the offline application, the options are generated through some random tasks beforehand and the experiments have shown that options can accelerate the primitive Q-learning greatly. The online application forms options autonomically during the learning process. The experiments have also shown that with the help of options, Q-learning achieves the optimal policies much faster.

This article also validated three minor problems in the UDV approach. We will further examine the approach in some moderately large problems in the future research. Additionally, although the UDV approach is limited to the grid-world environment, the action-restricted heuristics is widely applicable. The future research will consider its implementation in other environments, such as real life and dynamic task environments.

Acknowledgements

This work was supported by the National Basic Research Program of China (2013CB329603), the National Natural Science Foundation of China (61375058, 71231002), the China Mobile Research Fund (MCM 20130351), the Ministry of Education of China and the Special Co-Construction Project of Beijing Municipal Commission of Education.

References

1. Singh S P, Jaakkola T, Jordan M I. Reinforcement learning with soft state aggregation. Advance in Neural Information Processing Systems 7: Proceedings of the Neural Information Processing Systems Conference (NIPS'94), Nov 28–Dec 1, 1994, Denver, CO, USA. Cambridge, MA, USA: MIT Press, 1995: 361–368
2. Tsitsiklis J N, Van Roy B. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, 1997, 42(5): 674–690
3. Dietterich T G. Hierarchical reinforcement learning with the max Q value function decomposition. Journal of Artificial Intelligence Research, 2000, 13: 227–303
4. Parr R. Hierarchical control and learning for Markov decision processes. PhD Thesis. Berkeley, CA, USA: University of California, Berkeley, 1998
5. Simsek Ö, Wolfe P A, Barto A G. Identifying useful subgoals in reinforcement learning by local graph partitioning. Proceedings of the 22nd International Conference on Machine Learning (ICML'05), Aug 7–10, 2005. Bonn, Germany. New York, NY, USA: ACM, 2005: 816–823
6. Sutton R S, Precup D, Singh S. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 1999, 112(1/2): 181–211
7. Digney B L. Learning hierarchical control structure for multiple tasks and changing environments. From Animals to Animats 5: Proceedings of the 5th International Conference on Simulation of Adaptive Behavior (SAB'98). Aug 17–21, 1998, Zurich, Switzerland. Cambridge, MA, USA: MIT Press, 1998: 321–330
8. Mcgovern A, Barto A G. Automatic discovery of subgoals in reinforcement learning using diverse density. Proceedings of the 18th International Conference on Machine Learning (ICML'01), Jun 28–Jul 1, Williamstown, MA, USA. San Francisco, CA, USA: Morgan Kaufmann, 2001: 361–368
9. Stolle M, Precup D. Learning options in reinforcement learning. Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation (SARA'02), Aug 2–4, Kananaskis, Canada. Berlin, Germany: Springer, 2002: 212–223
10. Asadi M, Huber M. Autonomous subgoal discovery and hierarchical abstraction for reinforcement learning using Monte Carlo method. Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI'05), Jul 9–13, 2005, Pittsburgh, PA, USA. Cambridge, MA, USA: MIT Press, 2005: 1588–1589
11. Goel S, Huber M. Subgoal discovery for hierarchical reinforcement learning using learnt policies. Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS'03), May 12–14, 2003, St Augustine, FL, USA. 2003: 346–350
12. Mannor S, Menache I, Hoze I, et al. Dynamic abstraction in reinforcement learning via clustering. Proceedings of the 21st International Conference on Machine Learning (ICML'04), Jul 4–8, 2004, Banff, Canada. San Francisco, CA, USA: Morgan Kaufmann, 2004: 560–567
13. Menache I, Mannor S, Shimkin N. Q-cut-dynamic discovery of subgoals in reinforcement learning. Proceedings of the 13th European Conference on Machine Learning (ECML'02), Aug 19–23, 2002, Helsinki, Finland. Berlin, Germany: Springer, 2002: 295–306
14. Jing S, Gu G C, Liu H B. Automatic option generation in hierarchical reinforcement learning via immune clustering. Proceedings of the 1st International Symposium on Systems and Control in Aerospace and Astronautics(SSCAA'06), Jan 19–21, 2006, Harbin, China. Piscataway, NJ, USA: IEEE, 2006: 4p
15. Simsek Ö, Barto A G. Skill characterization based on betweenness. Advances in Neural Information Processing Systems 21: Proceedings of the 22 Annual Conference on Neural Information Processing Systems (NIPS'09), Dec 8–11, 2008, Vancouver, Canada. Cambridge, MA, USA: MIT Press, 2009: 1497–1504
16. Entezari N, Shiri M E, Moradi P. Subgoal discovery in reinforcement learning using local graph clustering. International Journal of Future Generation Communication and Networking, 2011,4(3): 13–23
17. He R J, Brunskill E, Roy N. PUMA: Planning under uncertainty with macro-actions. Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10), Jul 11–15, 2010, Atlanta, GA, USA. Cambridge, MA, USA: MIT Press, 2010: 1089–1096
18. Konidaris G, Barto A. Efficient skill learning using abstraction selection. Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09}, Jul 11–17, 2009, Pasadena, CA, USA. 2009: 1107–1113
19. Wang B N, Gao Y, Chen Z Q, et al. K-cluster subgoal discovery algorithm for option. Journal of Computer Research and Development, 2006, 42(5): 851–855 (in Chinese)
20. Sutton R S, Barto A G. Reinforcement learning: An introduction. Cambridge, MA, USA: MIT Press, 1998
21. Precup D. Temporal abstraction in reinforcement learning. Ph. D Thesis. Amherst, MA, USA: University of Massachusetts, 2000

(Editor: WANG Xu-ying)