Computer Science and Information Systems 13(2):561–577 DOI: 10.2298/CSIS160210015F

Learn to Human-level Control in Dynamic Environment Using Incremental Batch Interrupting Temporal Abstraction

Yuchen Fu[1], Zhipeng Xu[1], Fei Zhu[1,2,3,4], Quan Liu[1,3], and Xiaoke Zhou[5]

[1] School of Computer Science and Technology, Soochow University, Shizi Street No.1 Box 158, Suzhou, China, 215006
yuchenfu@suda.edu.cn,
20134227052@stu.suda.edu.cn,
{zhufei, quanliu}@suda.edu.cn
[2] Provincial Key Laboratory for Computer Information Processing Technology Soochow University
[3] Collaborative Innovation Center of Novel Software Technology and Industrialization
[4] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun, 130012, P.R. China
[5] University of Basque Country, Spanish
xzhou001@ikasle.ehu.eus

Abstract. The temporal world is characterized by dynamic and variance. A lot of machine learning algorithms are difficult to be applied to practical control applications directly, while hierarchical reinforcement learning can be used to deal with them. Meanwhile, it is a commonplace to have some partial solutions available, called options, which are learned from knowledge or predefined by the system, to solve sub-tasks of the problem. The option can be reused for policy determination in control. Many traditional semi-Markov decision process methods take advantage of it. But most of them treat the option as a primitive object. However, due to the uncertainty and variability of the environment, they are unable to deal with real world control problems effectively. Based on the idea of interrupting option under the prerequisite for dynamic environment, a Q-learning control method which uses temporal abstraction, named as I-QOption, is introduced. The I-QOption approach combines the idea of interruption with the characteristics of dynamic environment so as to be able to learn and improve control policy in dynamic environment. The Q-learning framework helps to learn from interaction with raw data and achieving human-level control. The I-QOption algorithm is applied to grid world, a benchmark dynamic environment evaluation testing. The experiment results show that the proposed algorithm can learn and improve policy effectively in dynamic environment.

Keywords: hierarchical reinforcement learning, option, reinforcement learning, online learning, dynamic environment.

1. Introduction

Reinforcement learning provides a framework to learn directly from the interaction and achieve goals. Reinforcement learning has been widely studied due to its great ability

of generalization. The reinforcement learning algorithm take advantage of reinforcement learning agent to constantly interact with the unknown environment during the process of solving the problem [2],[31]. As an important algorithm of reinforcement learning, the temporal difference (TD) learning is capable of learning directly from raw experience without determining dynamic model of environment in advance. Moreover, the model learned by temporal difference is updated by estimation which is based on part of learning rather than final results of the learning. TD learning is particularly suitable for solving the prediction problems and control problems in real-time control applications. The Q-learning, which is an off-policy version of TD learning, is capable of reducing the computational complex, and achieving human-level control.

However, as the scale of the problem to be solved gets larger and larger, the reinforcement learning agent requires more and more time, computation and information to learning and make decisions. As a result,the agent usually fails to work because it cannot handle mountains of data efficiently [14],[29]. Therefore, it seems to be essential to learn knowledge by interacting with the environment so as to attain a better policy for temporal decision making [20],[28]. As it has been revealed that the world is organized by some structures which contain information to help to make decision, we anticipate that the agent requires less computation by fully taking advantage of the structure of environment [13]. Botvinick et al. uses hierarchical reinforcement learning to make decisions [4]. Furthermore, the hierarchical reinforcement learning methods proposed by Bakker et al. made use of sub-goal discovery and sub policy specialization [1]. More recently, Jardim et al. [10] extended the abstraction to state and then put forward an algorithm to learn sub-goals and sub-states in the hierarchical reinforcement learning. The option frameworks reviewed by Barto et al. [3], which were based on temporal abstraction proposed by Sutton et al. [27], have been used in many systems. Although options provides a very useful framework for temporal abstraction, most work used sub-task to represent options [30],[11], correspondingly, many work aimed to find the sub-tasks [7]. For example, McGovern et al. used diverse density to automatically find sub-goals of reinforcement learning tasks [17]. Menache et al. put forward a Q-cut-dynamic method to discovery sub-goals [18]; Şimşek et al. introduced a method which used relative novelty to identify useful temporal abstraction [23] and a method which utilized local graph partitioning [24]. There are also lots of other work to extract options such as the value function [6], bisimulation metrics [5], visit frequency [26], via clustering[16], Using Ant System [8] and graph theory based method [12],[22].

An available temporal abstraction option is very critical to solve the problem efficiently [19]. The work [28] have had a preliminary study on it. However, his research was based on the setting of an ordinary simple condition, and therefore can not be applied in the real dynamic environment. Hence, in this work, we introduce the idea of interrupting to solve learning and controlling problems in dynamic environments, which brings two advantages: extending the ability of agent to solve the problems by introducing the idea of interrupting, which traditional option-based methods cannot deal with, and reducing the efforts by using SMDP methods.

This paper is organized as follows. After a brief introduction to reinforcement learning, options and SMDP framework in Sect. 2. We describe our algorithm and the dynamic environment in which we applied our algorithm. Finally, we use a few reinforcement learning tasks [17] to illustrate the usefulness of our algorithm.

2. Related Work

2.1. Reinforcement Learning

Reinforcement learning is a kind of machine learning method by interacting with the environment and mapping the states to the actions. In reinforcement learning, the agent evaluates the quality of actions by reward function and then takes the action that brings about the maximum returns, and thus the action will not only affect the immediate reward but also affect the reward of the next step, which is also known as reward of the next state [15]. Trial and error search as well as delayed reward are the important features of reinforcement learning. The reinforcement learning framework has five fundamental elements: controller, environment, state, reward, and action, showed as Fig. 1. The controller, which learns knowledge by interacting with outside environment and then chooses an action in accordance with the decision made by established controlling model, is the agent of the system; accordingly, the state will then be changed; the environment will return a reward for evaluating. Most of reinforcement learning methods are based on Markov De-

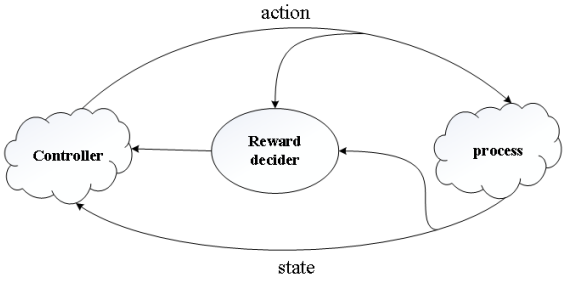


Fig. 1. The Framework of reinforcement learning.

cision Process (MDP) which is represented by a tuple $< X, U, f, \rho, \gamma >$, where X and U respectively represent a finite set of states and a set of actions, $f \in [0, 1]$ represents transition probability, $\rho : X \times U \to \Re$ denotes the rewards received by agent, and $\gamma \in [0, 1]$ represents discounted factor. At each time step, agent observes system state $x \in X$ then takes an action, and then the state of the system transfers to next state $x' \in X$ with probability$f(x, u, x')$, and agent will get an immediate reward. The goal of the agent is to find the optimal policy $h^* : X \times U \to [0, 1]$ through maximizing the cumulative expected reward [25].

At each time step t, the agent selects an action $u_t \in U$ from action space U. As a result, the state of the system transfers to $x_{t+1} \in X$ from $x_t \in X$ in accordance with a function of transition probability $f(x_t, u_t, x_{t+1})$:

$$x_{t+1} = f(x_t, u_t) \tag{1}$$

The agent attains a reward r_{t+1} according to rewarding function ρ:

$$r_{t+1} = \rho(x_t, u_t) \tag{2}$$

The state-action value function $Q^h : X \times U \rightarrow \Re$ under policy h is denoted as follows:

$$Q^h(x,u) = \sum_{t=0}^{\infty} \gamma_t \rho(x_t, u_t) \tag{3}$$

where $\gamma \in [0,1]$ is a discount rate which shows how far sighted the agent is in considering the rewards and is also a factor for increasing uncertainty on future rewards.

TD is capable of learning directly from raw experience without determining dynamic model of environment in advance. Moreover, the model learned by temporal difference is updated by estimation which is based on part of learning rather than final results. These two characteristics of TD make it particularly suitable for solving the prediction and control problems in real-time control applications. Given some experience with policy h, temporal difference learning updates estimated V-value function as:

$$V(x_t) \leftarrow V(x_t) + \alpha[R_t - V(x_t)] \tag{4}$$

where R_t is actual return after time step t, α is a step size parameter. TD estimates $V(_t)$ in step $t+1$ using the observed reward r_{t+1}.

Let $Q^h(x,u)$ be the value of taking action $u \in U$ under a policy h. $Q^h(x,u)$ is defined as:

$$Q^h(x,u) = \rho(x,u) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, u_k) \tag{5}$$

Q-learning is an off-policy version of TD method, which is defined by:

$$Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha[r_{t+1} + \gamma \max_u Q(x_{t+1},u) - Q(x_t,u_t)] \tag{6}$$

The ultimate goal of reinforcement learning agent is to get an optimal policy h^*. The corresponding optimal value function $V^*(x)$ and state-action value function $Q^*(x,u)$ can be represented as:

$$V^*(x) = \max_{u \in U} \{\rho(x,u) + \gamma \sum_{x \in X} f(x,u,x') V^*(x')\} \tag{7}$$

$$Q^*(x,u) = \rho(x,u) + \gamma \sum_{x \in X} f(x,u,x') \max_{u \in U} Q^*(x',u') \tag{8}$$

2.2. Options

In this work, we use Markov option [31] to represent the terms of temporal abstraction. Temporal abstractions and primitive actions are both actions selected by the agent. During the execution of an option, agent takes policy h of the option until the terminal condition of the option is satisfied. Generally, an option can be modeled by a triplet $< I, h, \beta >$. The input set of option is expressed by I, $I \in X$, which means an option $< I, h, \beta >$ is available at $x \in I$. Policy $h : X \times U \rightarrow [0,1]$ denotes the internal policy of o and terminal condition $\beta : X \rightarrow [0,1]$.

The implementation process of Markov option is as follows: if agent chooses option o at state x_t, then agent will choose next action according to the policy of option o, that is

$u_t \leftarrow h(x_t)$; and the state of the system will transfer to x_{t+1}, namely $(x_t, h) \xrightarrow{u_t} x_{t+1}$. Agent will determine whether to end the execution of o at x_{t+1} according to the terminal condition β. If the execution is not interrupted, o will continue to be executed until a terminal condition $\beta(x_{t+k}) \rightarrow 1$ is satisfied. So there exists $\beta(x) \rightarrow 0$ for all primitive actions. When an option terminates, agent could choose another option or choose a primitive action. We will use an example to illustrate option. A robot is required to open a door. The whole process is composed of inserting the key, holding the lock, rotating the lock, and opening the door. If we use the concept of option, the process can be regarded as an action sequence: inserting the key, holding the lock, and rotating the lock. During this process, agent standing by the door can starts with an initial state of option o. The next action executed by the agent could be inserting the key according the o's policy and then get to the next state recursively. The terminal state of the system should be that rotating the lock and o is terminated at the time. Consequently the agent will then select the next option or primitive action. In fact, the option also includes primitives actions, where all the primitive actions meet $\beta(x) \rightarrow 0$.

Now we can define policy on option. Let the available option set be O_x at x_t, when the agent starts from state x_t, an option o will be chosen with probability of $v(x_t, o)$, where $o \in O_x$ and v is a Markov policy. Then actions will be selected according to policy h of option o until o is terminated at state x_{t+k} after k steps; and then the next option o' will be selected according $v(x_{t+k})$. Actually, the policy v defined over option o determined the flat policy y defined on primitive actions, and we can get $y = f(v)$. So we can define the value of a state under the flat policy at state x_t:

$$V^y(x_t) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | \varepsilon(y, x_t, t)\} \tag{9}$$

where $\varepsilon(y, x_t, t)$ denotes the history that agent start from state x_t at time t under policy y. Due to this formula is based on the primitive actions, but policy y is determined by v, so there exist $V^y(x_t) = V^{f(v)}(x_t)$. Similarly we can get:

$$Q^v(x_t, o) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | \varepsilon(vo, x_t, t)\} \tag{10}$$

where $\varepsilon(yo, x_t, t)$ denotes the process that agent select option o first under policy v until o is terminated and select other options.

2.3. Semi-Markov Decision Problems (SMDPs)

A reinforcement learning task which satisfies the Markov property is considered as Markov decision processes (MDPs). We believe that a semi-Markov Decision Process (SMDP) can be constituted by any MDP and a fixed set of options [27]. Cnventional SMDP theory is associated with the actions and related methods can be extended to options [21]. Thus, for any option o, if $\varepsilon(o, x_t, t)$ denotes the process that o starts from state x_t at time t, then the corresponding reward model will be:

$$\rho(x_t, o) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{k-1} r_{t+k} | \varepsilon(o, x_t, t)\} \tag{11}$$

where $t + k$ represent termination time of option o. Similarly, the transition probability model will be defined as follows:

$$f(x_t, o, x_{t+k}) = \sum_{k=1}^{\infty} f(x_{t+k}, k)\gamma^k \tag{12}$$

where $f(x_{t+k}, k)$ denotes the probability that o terminates after k steps at state x_{t+k}. Then according to Bellman equation, for any Markov policy h, the state value function can be defined as:

$$V^h(x_t) = \sum\nolimits_{o \in O_{x_t}} h(x_t, o)[\rho(x_t, o) + \sum\nolimits_{x_{t+k}} f(x_t, o, x_{t+k}) V^u(x_{t+k})] \tag{13}$$

Corresponding state-action value function can be defined as:

$$Q^h(x_t, o) = \rho(x_t, o) + \sum\nolimits_{x_{t+k}} f(x_t, o, x_{t+k}) V^h(x_{t+k}) \tag{14}$$

On the basis of the value function, we can get the optimal value function. In MDPs, we select optimal action, correspondingly, we select optimal option. We denote options set by O here. According to Bellman optimal equation, we can obtain the optimal state value function:

$$V^*(x_t) = \max\nolimits_{o \in O_{x_t}} \rho(x_t, o) + \sum\nolimits_{x_{t+k}} f(x_t, o, x_{t+k}) V^*(x_{t+k}) \tag{15}$$

And the optimal state-action value functions:

$$\begin{aligned} Q^*(x, o) &= \rho(x_t, o) + \sum\nolimits_{x_{t+k}} f(x_t, o, x_{t+k}) V^*(x_{t+k}) \\ &= \rho(x_t, o) + \sum\nolimits_{x_{t+k}} f(x_t, o, x_{t+k}) \max\nolimits_{o' \in O_{x_{t+k}}} Q^*(x_{t+k}, o') \end{aligned} \tag{16}$$

According to the optimal value function, we can get updating formula of Q:

$$Q^h(x_t, o) \leftarrow Q^h(x_t, o) + \alpha[\rho(x_t, o) + \gamma^k \max_{o'} Q^h(x_{t+k}, o') - Q^h(x_t, o)] \tag{17}$$

If the option set has already been obtained, then we can compute the optimal state value function and the optimal action value function. Finally we can obtain the optimal policy through interacting with the environment. Moreover, the standard SMDP theory can guarantee that such process can converge.

3. Algorithm description

3.1. Interrupting Option

By using option, agent is more efficient in dealing with exploration tasks [9], so the algorithm can converge faster [27]. At the same time, knowledge can be reused in solving duplicate tasks when options are utilized [6],[30]. As a result, agent doesn't have to learn from scratch. Option is also introduced in SMDP methods. However, most traditional SMDP methods with option take option as an opaque indivisible action, which makes the agent clumsy. Therefore, we change the structure of the option. Here we consider using the interrupting options proposed in [28], that is, before option terminates according to its terminal condition, we could interrupt the execution of an option if there is a need. For example, in the room navigation task, we assume that if we treat the action sequences that agent from the door entrance to entering into the room as an option, when agent is executing the option and the moment that the agent just ready to step into the room. If the door is closed unexpectedly, according to the definition of traditional SMDP, the option will continue without considering termination, which is in fact the fit action as it is able to greatly reduce useless computation and thus improve the efficiency. Meanwhile, the problem is able to be easier to solved if we use the concept of interrupting options.

3.2. Incremental Batch Updating Approach

Batch reinforcement learning is often used to describe a set of reinforcement learning that complete learning from a group of samples, the key lies in batch mode algorithm is the way it processes a batch of sample and gets the best results from it. The benefits of batch updating are the stability of the learning process and the validity of the data, and reinforcement learning methods using batch updating usually converge faster to a certain extent. By using batch reinforcement learning all of the observed transition samples are stored and synchronous updating on the entire sample. However, if it is in accordance with the first storing all the samples and then updating to learn a good policy, policy of reinforcement learning agent will not be updated during sampling. In practice, sampling has a great influence on the quality of policy. At the same time, the samples for reinforcement learning agent must be similar to the actual transition sample of the system in order to get a good policy. The simplest way to solve it is to interact with the system to sample, which gives rise to incremental batch updating methods between online learning methods and batch updating methods. The process of incremental reinforcement learning methods for batch updating learning is shown in Fig. 2.

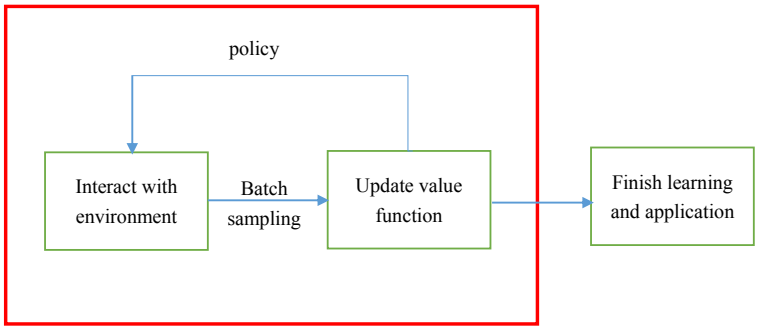


Fig. 2. The process of incremental batch reinforcement learning methods.

3.3. Algorithm I-QOption

In traditional reinforcement learning algorithm, agent learns value function and policy through interacting with the environment constantly. However in the environment of large scale, agent requires a lot of time and knowledge to interact with the environment in order to obtain a better policy. In this work, we utilize the SMDP to model the system and use the option to improve the efficiency of exploration the environment so as to speed up the convergence and ensure the stability of the early learning algorithm performance. Conventional SMDP methods take option as a whole and rather than a fissile one such that once the option is getting started, the agent must carry on the the option until it is

finished without any termination halfway. As a matter of fact, this approach is known to be confronted with the problem in both cases. Firstly, the efficiency will be very poor in a dynamic environment as usually the option has been stalled before it ends as planned; secondly during the execution of the option, in certain states, it would be better to select another option. In this case, our algorithm should be applied to interrupt current option at appropriate time.

Suppose we've got the value function of an option under policy h, where h is a global policy and (x, o) is a state option pair. The state-option value function $Q^h(x, o)$ cannot only evaluate the quality of current policy h, but also evaluate the quality of every step we take. Assume that at time step t, according to policy h, agent has selected option o, then we can compare the value that agent execute o with the value that interrupt o and select new option:

$$V^h(x) = \sum_{o'} u(x, o')Q^h(x, o')$$

If $V^h(x_t) > Q^h(x_t, o)$ or $Q^h(x_t, o_t) < Q^h(x_{t+1}, o_t)$ when $x_{t+1} = x_t$, we will interrupt o and complete an incremental batch updating, and then select another option to continue. The description of the algorithm is given as follows.

Algorithm 1 I-QOption

Require: discounted factor γ, learning rate α, option set O_g

Initialize $Q(x, o)$ arbitrarily,where $o \in O$
repeat
  Initialize x_t
  repeat
    Select an option o(according to the initial exploration strategy h)
    Execute option o
    repeat
      Select action u (according to $h_o(x_t)$)
      Observe next state and reward x_{t+1}, r
      Save x_t, x_{t+1}, r
      $x_t \leftarrow x_{t+1}$
      if $Q^h(x_t, o_t) < Q^h(x_{t+1}, o_t)$ when $x_{t+1} = x_t$ or $V^h(x_t) > Q^h(x_t, o)$ or $\beta(x_{t+1}) = 1$ **then**
        Batch updating $Q^h(x_t, o)$ for every x_t in o
        $\beta(x_{t+1}) = 1$
      end if
    until o ended
    Select new option o' according to $Q(x_t, o)$
  until x_{t+1} is the terminal state
until convergence

3.4. Algorithm Analysis

Definition 1. *For any MDP, any option set O and arbitrary Markov policy h, we define a new option set O′, there is an one to one mapping between the two option sets: we*

denote a corresponding $o' \in O$ for every $o =< I, h_o, \beta >\in O$, where $\beta = \beta'$ except $Q^h(h_o, o) < V^h(x)$, H represents history and x represents the last state of history H. We will terminate o' at state x, namely $\beta'(x) = 1$. All of the history interrupt like this is called interrupting history.

Theorem 1. *Let h' over o' to be the corresponding policy of $h : h(x, o) = h'(x, o')$, then: 1.$V^{h'}(x) \geq V^h(x)$ for all $x \in X$; 2.there is a non-zero probability to encounter interrupting history if initialized from state $x \in X$, then there is $V^{h'}(x) > V^h(x)$; 3.$\lim_{k\to\infty} V_k(x) = V_o^*(x)$ for all of $x \in X, o \in O$, that is the algorithm can converge to a fixed point.*

Proof. The idea is to execute improved policy h' by improving its terminal condition for any state x, that is, we need to prove that the following inequality is satisfied

$$\sum\nolimits_{o'} h'(x, o')[\rho(x, o') + \sum\nolimits_{x'} f(x, o', x')V^h(x')] \geq V^h(x) \tag{18}$$

where $V^h(x) = \sum_o h(x, o)[\rho(x, o) + \sum_{x'} f(x, o', x')V^h(x')]$, if inequality (18) is satisfied, we can use it to extend the left part by using

$$\sum\nolimits_{o'} h'(x, o')[\rho(x, o') + \sum\nolimits_{x'} f(x, o', x')V^h(x')]$$

constantly. In the limit case, the left formula turns to be V^h. Then we can get $V^{h'} \geq V^h$.

Because of $h'(x, o') = h(x, o), \forall x \in X$, we need to proof :

$$\rho(x, o') + \sum\nolimits_{x'} f(x, o', x')V^u(s') \geq \rho(x, o) + \sum\nolimits_{x'} f(x, o, x')V^h(x') \tag{19}$$

Let Φ denote all the interrupting history $\Phi = \{H \in \Omega : \beta(H) \neq \beta'(H)\}$, then the left side of inequality (19) can be rewritten as:

$$E\{r + \gamma^k V^h(x')|\epsilon(o', x), H_{xx'} \notin \Phi\} + E\{r + \gamma^k V^h(x')|\epsilon(o', x), H_{xx'} \in \Phi\} \tag{20}$$

where x' denotes the next state, r denotes the immediate reward and k denotes step number after following option o from state x respectively. The history from state x to x' is denoted by x'. Due to encounter the trajectory $H_{xx'} \notin \Phi$, so the trajectory will be terminated, and it will be occur with the same probability and expectation after execute o at state x. Therefore, the right side of the inequality (19) can be rewritten as:

$$\begin{aligned} &E\{r + \gamma^k V^h(x')|\epsilon(o', x), H_{xx'} \notin \Phi\}+ \\ &E\{\beta(x')[r + \gamma^k V^h(x')] + (1 - \beta(x'))[r+ \\ &\gamma^k Q^h(H_{xx'}, o)]|\epsilon(o', x), h_{xx'} \in \Phi\} \end{aligned} \tag{21}$$

Because $Q_o^h(H_{xx'}, o) \leq V^x(x')$ for any history $H_{xx'} \in \Phi$, so inequality (18) is proofed which is $V^{h'} \geq V^h$. If there exist at least one history of trajectory produced by o' with probability non-zero, then the inequality strictly holds, that is $V^{h'} > V^h$

4. Experiment and Results

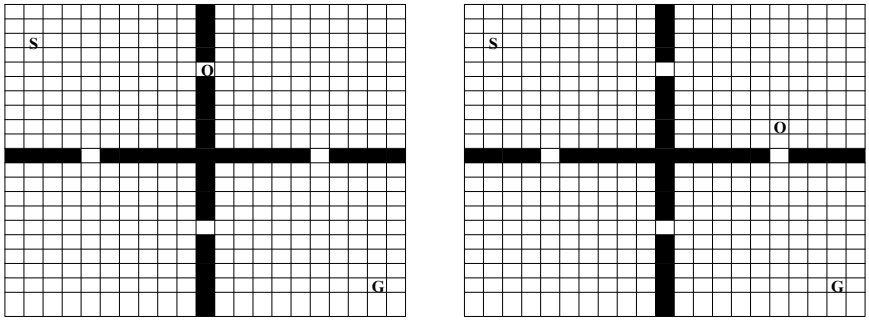
As the uncertainty of the environment in the real world, conventional SMDP methods applied to options cannot be utilized efficiently, we use the option in a dynamic environment here to solve the task better. We will use the famous gridworld simulation experiment to evaluate the behaviour of I-QOption compared with Q-learning, and then we will give the experimental results below.

In the simulation experiment, agent uses the $\varepsilon - greedy$ policy to complete exploration, the initial exploration probability is set to be $\varepsilon = 0.1$ and the learning rate is set to be $\alpha = 0.1$. In order to let the algorithm to get a better convergence, the exploration probability will decay with the increase of the number of the episode. Here we set $\varepsilon \leftarrow \varepsilon/episode$. All of the Q value will be initialized to 0.

4.1. Dynamic Environment Description

So far, most of the reinforcement learning methods are applied to some simple learning task or learning in a static environment, such as balancing pole, DC motor, roller coaster, etc. However, real-world environment is usually not static. For example, in the room navigation task, there is no obstacle besides wall in general environment, even if there are obstacles, the position of those obstacles will not change over time. In real-world room navigation task, the random appearance of obstacles are very normal. Our target is to find optimal policy in a dynamic, ever-changing environment.

Fig. 3 gives an example of dynamic environment, it is a grid of 21×21. There are two kinds of objects in this environment: agent and obstacles. The uppercase letter o in the left lattice in Fig. 3(a) indicates the position of the obstacle at episode t while the letter o in the right lattice in Fig. 3(b) indicates the position of the obstacle in episode $t + k$. As can be seen that in a different episode the location of the obstacle may not be the same.

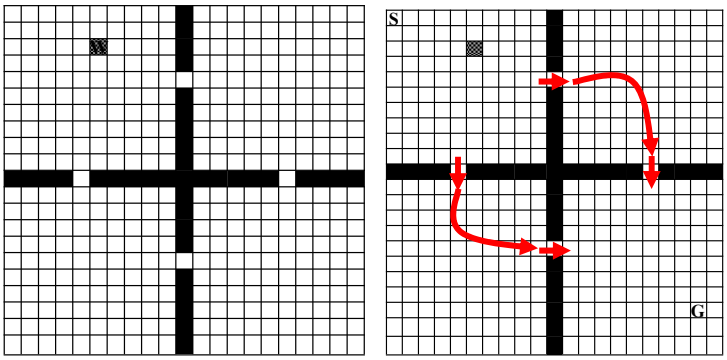


(a) (b)

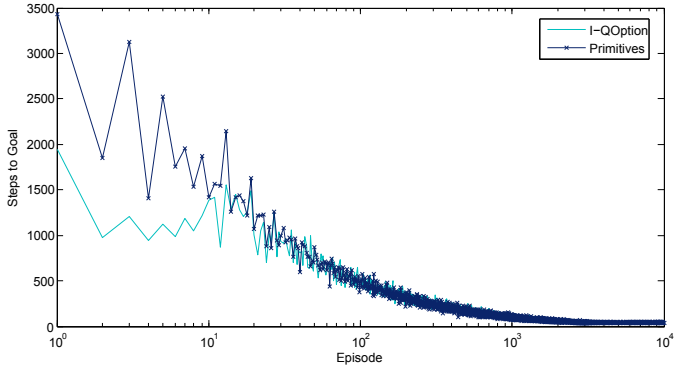
Fig. 3. The dynamic environment of a 21×21 grid where S is the start position, G is the target position and O is the obstacle. The difference between the two figures is the position of O, which means that in different episode, agent will be in different situation even if the position of the agent is the same, because the obstacle set O has changed.

4.2. Four-Room Dynamic Gridworld

We give a four-room dynamic gridworld environment shown in Fig. 4(a). The goal state G is placed in the lower right corner and the initial state S is placed in the upper left corner. At the beginning of each episode, we will provide random obstacles, besides the fixed obstacles, to represent the random environment. The primitive action consists of four direction actions: up, down, left and right. The selection probability of greedy action (primitive action or option) is $1-\varepsilon+\varepsilon/(|U|_x+|O|_x)$, and the selection probability of other action or option is $\varepsilon/(|U|_x+|O|_x)$. The reward is set to be -1 except the step to the goal state which has been set to be 1.



(a) 4-room dynamic environment (b) Pre-defined options



(c) Comparison of experimental results

Fig. 4. Comparison of I-QOption algorithm and Q-learning in four-room dynamic environment.

Since the focus of this paper is the application of option in dynamic environments, therefore, the options here are predefined. We provide six options as prior knowledge in the four-room gridworld shown in 4(b).

Fig. 4(c) shows the average step number, agent from the initial state to the goal state based on 10 repeated experiments, which compares the performance of I-QOption and

Q-learning in the dynamic gridworld. As can be seen from Fig. 4(c), the step I-QOption needs to reach the goal state is far less than Q-learning in the first few episodes. In the first episode I-QOption needs only 1829 steps while Q-learning needs 2542 steps. The Fig. 4(c) also shows that the convergence rate of I-QOption is slightly faster than Q-learning. I-QOption converges at the episode 3708 while Q-learning converges at episode 3787.

As can be seen from Fig. 4(c), in the four-room dynamic environment, the performance of I-QOption is significantly better than Q-learning which based on primitive actions in the first few episodes, and this phenomenon is particularly evident in the first 10 episodes. After 10 episodes, the performance of I-QOption and Q-learning is roughly equal, but I-QOption still has a faster convergence rate than Q-learning for about 80 episodes. The simulation results indicate the effectiveness of the algorithm in dynamic environments.

4.3. Six-Room Dynamic Gridworld

In the second experiment described in this paper, we use six-room dynamic gridworld to simulate experiments. The task of the agent is the same as that of four-room dynamic gridworld. Fig. 5(a) shows the experimental environment, the initial state close to the upper left corner while the goal state near the lower right corner. Dynamic environment and primitive actions are set to be the same as introduced before.

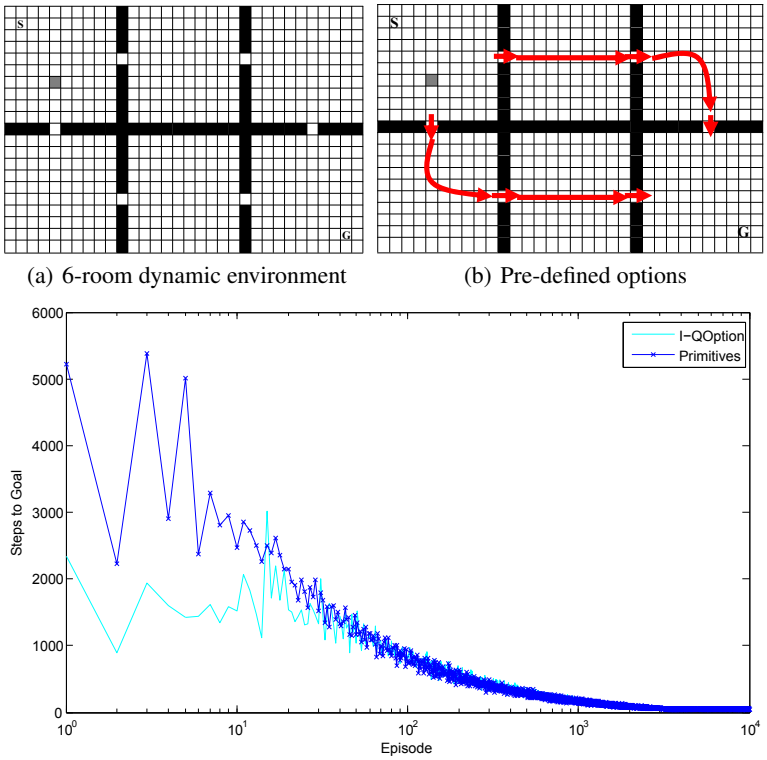
As described in the previous section, the number of option pre-defined here will be a corresponding increase due to the number of room increased. In the six-room experiment, 10 options will be pre-defined while six options are provided in the four-room experiment as Fig. 5(b) shown. Fig. 5(c) shows the average step number that agent from the start state to reach the goal state, based on 10 repeated experiments. The difference with Fig. 4(c) is that the number of steps that agent needs to reach the goal state at early moment has increased due to the increasing of the state number. At the same time, the convergence rate also has slowed down, but the trend is the same overall.

We can draw the similar conclusion with the experiment results in the previous section, in the early stages of learning, I-QOption is much better than Q-learning, and in the latter study, the convergence rate of I-QOption is slightly better than Q-learning.

4.4. Results by Different Parameters Settings

There are three parameters in our algorithm: step size α, exploring probability ε, and discounted factor γ, where ε and γ usually have a general value, such as $\varepsilon = 0.1$.

The Table 1 shows learning result with different learning rate. In the case of learning rate a = 0.1, I-QOption only requires 1855 time steps to get to the goal state in the first episode while Q - learning needs 3827 time steps. It indicates that I-QOption has more than twice high efficiency of Q-learning. In the 10th episode, I-QOption becomes more efficient and only requires 1064 time steps to get to the goal, while Q-learning still needes 1485 time steps. It shows that in both algorithms, the previous knowledge is fully utilized so that it needs less time steps to complete carrying out the task during episode iterations. At the same time, it also shows that even at the beginning of the learning phase, the I - QOption algorithm, which is based on the interrupt mechanism, surpasses Q-learning in learning efficiency.



(a) 6-room dynamic environment (b) Pre-defined options

(c) Comparison of experimental results

Fig. 5. Comparison of I-QOption algorithm and Q-learning in six-room dynamic environment.

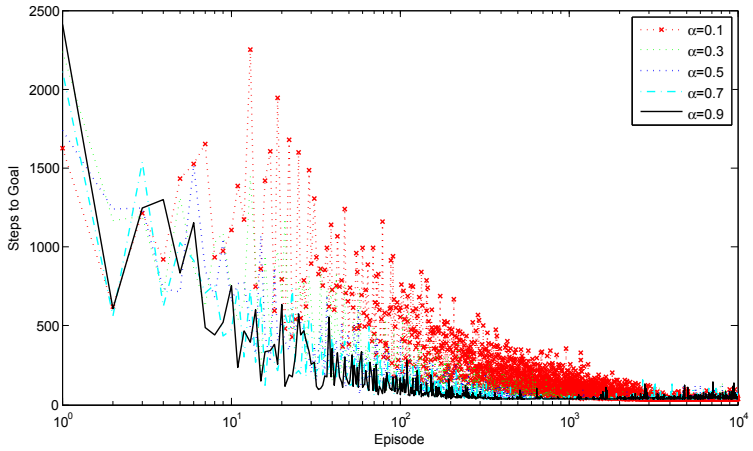
In the case of learning rate a = 0.9, I-QOption only requires 1687 time steps to reach goal in the first episode, while Q-learning needes 2995 time steps. The results suggest that we can accelerate the convergence by increasing the learning rate when the rate is within the acceptable range.

As can be seen from Fig. 6, both in the four-room and six-room experiments, the effects of different values of the algorithm are great. We can conclude that the learning speed of the agent has accelerated with the increase of α. Agent learns faster than other parameter values when $\alpha = 0.9$, and the performance is better in the learning phase. The proposed algorithm converges in the eightieth episode and maintain stability when $\alpha = 0.9$.

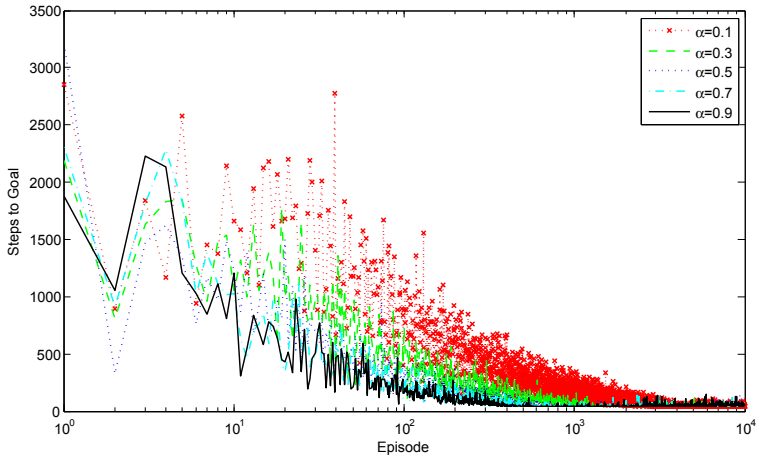
We can also see from Fig. 6(b) that, the larger value of α results in fast convergence speed of I-QOption. The convergence speed of algorithm with $\alpha = 0.1$ is much slower than that with $\alpha = 0.9$; and with more episodes for iteration, the fluctuation range of the algorithm is much smaller than that with $\alpha = 0.9$. When the algorithm converges to a certain extent, the performance of algorithm with $\alpha = 0.1$ appears to be more steady. As for convergence speed, the parameter setting with $\alpha = 0.9$ is faster, which is because in

Table 1. Comparison of I-QOption algorithm and Q-learning with different parameters in the 4-room experiment.

algorithm	parameter	steps first episode needs	steps tenth episode needs	episode number convergence required
I-QOption	$\alpha = 0.1$	1855	1064	3201
	$\alpha = 0.9$	1687	555	398
Q-learning	$\alpha = 0.1$	3827	1485	3245
	$\alpha = 0.9$	2995	555	410



(a) Different α on learning speed in 4-room experiment



(b) Different α on learning speed in 6-room experiment

Fig. 6. Effects of different parameters on the I-QOption algorithm.

the earlier learning phase, the value of $Q(x, u)$ with a large value of α tends to update quickly, which leads to the algorithm convergence in a short period of time. But because of the exploration of our algorithm, after the convergence of the algorithm, the large value of α leads to a more updating on exploration actions, which will causes larger fluctuation range.

5. Conclusions

Reinforcement learning enables to learn and optimize control from the interaction with environment. It is flexible and can be applied in many different applications. As a famous algorithm of reinforcement learning, Q-learning is able to learn directly from raw experience like human and does not need to know the environment model in advance.

In this work we introduce interrupting mechanism to SMDP option on the basis of Q-learning to address dynamic environments. The algorithm, called interrupting Q-learning option, is able to better solve the problem in a dynamic environment which traditional option based methods are unable to deal with. The experiment results show that the appropriate use of interrupting option can accelerate solving tasks in a dynamic environment, and also it will be help for agent to keep stability during the process of learning.

Acknowledgments. This paper is supported by National Natural Science Foundation of China (61303108, 61373094, 61472262), High School Natural Foundation of Jiangsu(13KJB520020), Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University(93K172014K04), Suzhou Industrial application of basic research program part (SYG201422), Provincial Key Laboratory for Computer Information Processing Technology, Soochow University(KJS1524).

References

1. Bakker, B., Schmidhuber, J.: Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In: Proc. of the 8-th Conf. on Intelligent Autonomous Systems. pp. 438–445 (2004)
2. Barto, A.G.: Reinforcement learning: An introduction. MIT press (1998)
3. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems 13(4), 341–379 (2003)
4. Botvinick, M.M.: Hierarchical reinforcement learning and decision making. Current opinion in neurobiology 22(6), 956–962 (2012)
5. Castro, P.S., Precup, D.: Automatic construction of temporally extended actions for mdps using bisimulation metrics. In: Recent Advances in Reinforcement Learning, pp. 140–152. Springer (2012)
6. Chaganty, A.T., Gaur, P., Ravindran, B.: Learning in a small world. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 391–397. International Foundation for Autonomous Agents and Multiagent Systems (2012)
7. Drummond, C.: Accelerating reinforcement learning by composing solutions of automatically identified subtasks. Journal of Artificial Intelligence Research 16(1), 59–104 (2002)
8. Ghafoorian, M., Taghizadeh, N., Beigy, H.: Automatic abstraction in reinforcement learning using ant system algorithm. In: AAAI Spring Symposium: Lifelong Machine Learning (2013)
9. Girgin, S., Polat, F., Alhajj, R.: Improving reinforcement learning by using sequence trees. Machine Learning 81(3), 283–331 (2010)

10. Jardim, D., Nunes, L., Oliveira, S.: Hierarchical reinforcement learning: learning sub-goals and state-abstraction. In: Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on. pp. 1–4. IEEE (2011)
11. Jin, Z., Jin, J., Liu, W.: Autonomous discovery of subgoals using acyclic state trajectories. In: Information Computing and Applications, pp. 49–56. Springer (2010)
12. Kazemitabar, S.J., Beigy, H.: Automatic discovery of subgoals in reinforcement learning using strongly connected components. In: Advances in Neuro-Information Processing, pp. 829–834. Springer (2008)
13. Konidaris, G., Barto, A.G.: Building portable options: Skill transfer in reinforcement learning. In: IJCAI. vol. 7, pp. 895–900 (2007)
14. Korf, R.E.: Macro-operators: A weak method for learning. Artificial intelligence 26(1), 35–77 (1985)
15. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning 8(3-4), 293–321 (1992)
16. Mannor, S., Menache, I., Hoze, A., Klein, U.: Dynamic abstraction in reinforcement learning via clustering. In: Proceedings of the twenty-first international conference on Machine learning. p. 71. ACM (2004)
17. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. Computer Science Department Faculty Publication Series p. 8 (2001)
18. Menache, I., Mannor, S., Shimkin, N.: Q-cut-dynamic discovery of sub-goals in reinforcement learning. In: Machine Learning: ECML 2002, pp. 295–306. Springer (2002)
19. Osentoski, S., Mahadevan, S.: Basis function construction for hierarchical reinforcement learning. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 747–754. International Foundation for Autonomous Agents and Multiagent Systems (2010)
20. Precup, D.: Temporal abstraction in reinforcement learning (2000)
21. Precup, D., Sutton, R.S., Singh, S.: Multi-time models for temporally abstract planning. Advances in neural information processing systems pp. 1050–1056 (1998)
22. Şimşek, Ö., Barreto, A.S.: Skill characterization based on betweenness. In: Advances in neural information processing systems. pp. 1497–1504 (2009)
23. Şimşek, Ö., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning. p. 95. ACM (2004)
24. Şimşek, Ö., Wolfe, A.P., Barto, A.G.: Identifying useful subgoals in reinforcement learning by local graph partitioning. In: Proceedings of the 22nd international conference on Machine learning. pp. 816–823. ACM (2005)
25. Singh, S., Jaakkola, T., Littman, M.L., Szepesvári, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. Machine Learning 38(3), 287–308 (2000)
26. Stolle, M., Precup, D.: Learning options in reinforcement learning. In: Abstraction, Reformulation, and Approximation, pp. 212–223. Springer (2002)
27. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence 112(1), 181–211 (1999)
28. Sutton, R.S., Singh, S., Precup, D., Ravindran, B.: Improved switching among temporally abstract actions. Advances in neural information processing systems pp. 1066–1072 (1999)
29. Szepesvari, C., Sutton, R.S., Modayil, J., Bhatnagar, S., et al.: Universal option models. In: Advances in Neural Information Processing Systems. pp. 990–998 (2014)
30. Thrun, S., Schwartz, A., et al.: Finding structure in reinforcement learning. Advances in neural information processing systems pp. 385–392 (1995)
31. Wiering, M., van Otterlo, M.: Reinforcement learning. Adaptation, Learning, and Optimization 12 (2012)

Yuchen Fu is a member of China Computer Federation. He is a PhD and professor. His research interest covers reinforcement learning, intelligence information processing, and deep Web.

Zhipeng Xu is a postgraduate student in the Soochow University.His main research interests include reinforcement learning and machine learning.

Fei Zhu is a member of China Computer Federation. He is a PhD and an associate professor. His main research interests include reinforcement learning, image processing, and pattern recognition. He is the corresponding author of this paper.

Quan Liu is a member of China Computer Federation. He is a PhD, post-doctor, professor and PhD supervisor. His main research interests include reinforcement learning, intelligence information processing and automated reasoning.

Xiaoke Zhou is now an assistant professor of University of Basque Country UPV/EHU, Faculty of Science and Technology, Campus Bizkaia, Spain. He majors in computer science and technology. His main interests include machine learning, artificial intelligence and bioinformatics.

Received: February 10, 2016; Accepted: May 29, 2016.