Engineering Applications of Artificial Intelligence 26 (2013) 2184–2193

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Backward Q-learning: The combination of Sarsa algorithm and Q-learning

Yin-Hao Wang, Tzuu-Hseng S. Li*, Chih-Jui Lin

aiRobots Laboratory, Department of Electrical Engineering, National Cheng Kung University, 1 University Road, Tainan 70101, Taiwan, ROC

ARTICLE INFO

Article history:
Received 4 November 2012
Received in revised form
14 June 2013
Accepted 27 June 2013
Available online 12 August 2013

Keywords:
Backward Q-learning
Q-learning
Reinforcement learning
Sarsa algorithm

ABSTRACT

Reinforcement learning (RL) has been applied to many fields and applications, but there are still some dilemmas between exploration and exploitation strategy for action selection policy. The well-known areas of reinforcement learning are the Q-learning and the Sarsa algorithms, but they possess different characteristics. Generally speaking, the Sarsa algorithm has faster convergence characteristics, while the Q-learning algorithm has a better final performance. However, Sarsa algorithm is easily stuck in the local minimum and Q-learning needs longer time to learn. Most literatures investigated the action selection policy. Instead of studying an action selection strategy, this paper focuses on how to combine Q-learning with the Sarsa algorithm, and presents a new method, called backward Q-learning, which can be implemented in the Sarsa algorithm and Q-learning. The backward Q-learning algorithm directly tunes the *Q*-values, and then the *Q*-values will indirectly affect the action selection policy. Therefore, the proposed RL algorithms can enhance learning speed and improve final performance. Finally, three experimental results including cliff walk, mountain car, and cart–pole balancing control system are utilized to verify the feasibility and effectiveness of the proposed scheme. All the simulations illustrate that the backward Q-learning based RL algorithm outperforms the well-known Q-learning and the Sarsa algorithm.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Reinforcement learning (RL) algorithms (Kaelbling et al., 1996; Sutton and Barto, 1998; Sutton, 1988; Watkins and Dayan, 1992) are different from supervised learning. RL uses a scalar reinforcement signal or a reward to interact with a complex environment. RL maps situations to actions so as to maximize a numerical reward through systematic learning. In order to obtain its own experience, the RL agent offers a trade-off between exploration and exploitation, so it not only has to exploit what it already knows through greater action in the current experience, but also has to explore what action will work better in the future (Sutton and Barto, 1998). Therefore, how to balance the exploration and exploitation is an important challenge (Derhami et al., 2010; Guo et al., 2004; Hwang et al., 2004; Lin and Li, 2003; Wiering and Hasselt, 2007, 2008).

RL has become one of the machine learning algorithms since 1980s (Dong et al., 2008) and has been applied to a wide range of fields, such as multiagent systems (Hwang et al., 2004; Wang and de Silva, 2008), robotics (Boubertakh et al., 2010; Harandi et al., 2009; Kamio and Iba, 2005; Li et al., 2011; Millan, 1996; Sharma and Gopal, 2008; Wurm et al., 2010), and other applications (Abdi et al., 2012; Chu et al., 2008; Mori and Ishii, 2011; Rahimiyan and Mashhadi, 2010). In Hwang et al. (2004), an adaptive Q-learning that achieves multiagent cooperation and coordination was proposed to develop a self-learning cooperative strategy for robot soccer systems. Wang and de Silva (2008) integrated the RL and genetic algorithms based on the multiagent architecture, which is used to learn multi-robot coordination. In Li et al. (2011), a policy gradient RL (PGRL) was utilized to adjust the walking parameters for biped robots, which can execute real-time performance and directly modify the policy. Kamio and Iba (2005) proposed an integrated technique for genetic programming and RL to apply to a real robot in a real environment. Sharma and Gopal (2008) proposed fuzzy Markov games as an adaptation of fuzzy Q-learning (FQL). This method utilized the reinforcement signal to tune the online conclusion part of a fuzzy Markov game controller for a robot manipulator. In Boubertakh et al. (2010), the fuzzy logic-based navigation method was proposed for a robot in an unknown environment, and modified Q-learning was adopted to tune the fuzzy rule base parameters. In Millan (1996), RL was provided to make an autonomous robot avoid collision and acquire efficient navigation strategies in a few trials. In Wurm et al. (2010), RL was applied to select the model for an autonomous navigation mobile robot. Also, an RL signal was designed to correlate with the face recognition accuracy, which was proposed in Harandi et al. (2009). Chu et al. (2008) designed a controller to walk on a flat terrain based on passive dynamic walking for actuated robots.

* Corresponding author. Tel.: +886 6 2757575; fax: +886 6 2342789.
E-mail addresses: tslee1689@gmail.com, thsli@mail.ncku.edu.tw (T.-H. Li).

0952-1976/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.engappai.2013.06.016

An adaptive Q-learning algorithm was designed to model the power supplier's strategic behavior in the electricity market, which was proposed in Rahimiyan and Mashhadi (2010). Aissani et al. (2009) utilized Sarsa algorithm to solve the dynamic maintenance task scheduling for a petroleum industry production system. Besides, in order to overcome computational cost, Mori and Ishii (2011) proposed a scheme for automatic construction of basis functions in Markov decision processes (MDPs). Abdi et al. (2012) proposed an emotional temporal difference learning algorithm for the forecasting the traffic-flow.

Even though RL is a useful learning machine, RL still has some difficult problems in practical applications. One problem is balancing the exploration–exploitation dilemma, while the other is its slow learning speed (Derhami et al., 2010; Guo et al., 2004; Hwang et al., 2004; Lin and Li, 2003; Sutton and Barto, 1998; Wiering and Hasselt, 2007, 2008). Thus, these problems have been extensively studied through the action selection policy. Sutton and Barto (1998) used Boltzmann distribution or ε-greedy method for exploration policy, where the temperature factor T or ε gradually decreases with an increased number of episodes. In recent years, Wiering and Hasselt (2007, 2008) proposed four ensemble algorithms in RL. Their method combined with Q-learning, Sarsa algorithm, actor-critic (AC) (Sutton and Barto, 1998), Q- and V-functions (QV-learning), and AC learning automaton (ACLA) (Wiering and Hasselt, 2007) in a single agent, and integrated with policies derived from the value function of the different RL algorithms. Due to the integration of different RL algorithms, many parameters have to be tuned, which takes a lot of computational time.

Recently, the artificial intelligent techniques have been applied to many fields (Chau, 2007; Cheng and Chau, 2001; Muttil and Chau, 2006; Taormina et al., 2012; Wu et al., 2009; Zhang and Chau, 2009). For example, Muttil and Chau (2006) applied artificial neural networks and genetic programming to analyze the water quality data from Tolo Harbour. A novel multi-sub-swarm particle swarm optimization proposed in Zhang and Chau (2009) was used to find multi-solutions for the multilayer ensemble pruning model. Chau (2007) adopted a particle swarm optimization model to train perceptrons in predicting the outcome of construction claims in Hong Kong. Taormina et al. (2012) presented the simulation of feed forward neural networks for hourly ground water levels in a coastal unconfined aquifer sited in the Lagoon of Venice, Italy.

Bianchi et al. (2008) combined RL algorithms with heuristic functions to select an action in order to accelerate the rate of convergence. A modified Q-learning based on the Metropolis criterion of the simulated annealing algorithm (SA-Q-learning) was proposed in Guo et al. (2004) to balance the exploitation and exploration of Q-learning. SA-Q-learning converges more quickly than Q-learning or Boltzmann exploration. In Lin and Li (2003), feature-SARSA was presented as well as local state features to adjust the current exploration strategy. An enhanced fuzzy Sarsa learning (EFSL) (Derhami et al., 2010) was offered by integrating the adaptive learning rate and the fuzzy balancer into fuzzy Sarsa learning (FSL) (Derhami et al., 2008), in which overfitting of parameters to highly visited states was prevented by employing the adaptive learning rate, and proper exploitation–exploration balance management was achieved by using a fuzzy balancer.

The two most well-known RL algorithms are Q-learning and Sarsa algorithms. Q-learning (Sutton and Barto, 1998; Watkins and Dayan, 1992) used a reward or scalar reinforcement signal to realize a specific task, where an off-policy temporal-difference (TD) control algorithm was developed. A Sarsa algorithm (Sutton and Barto, 1998; Sutton, 1996) was proposed for a modified Q-learning algorithm, which was the evolution of an on-policy TD control algorithm. The Sarsa algorithm and Q-learning are different between on-policy and off-policy (Sutton and Barto, 1998). It is addressed in Sutton and Barto (1998) and Wiering and Hasselt (2008) that on-policy method guarantees the convergence, while Boyan and Moore (1995) and Gordon (1995) showed that Q-learning might be divergent. In general, the Sarsa algorithm has a faster convergence characteristic, while Q-learning gives a better final performance. However, most literature studied action selection policy of Q-learning or Sarsa algorithm for the sake of balancing the exploitation–exploration dilemma. In order to solve these problems, other feasible alternatives offered by some researchers. For example, Hwang et al. (2004) presented a method to dynamically regulate learning rate, discount rate, and temperature factor in Boltzmann distribution, because fixing three parameters were not suitable for learning system. The Metropolis criterion presented in Guo et al. (2004) was applied to the search procedure in order to control the balance between exploration and exploitation. Derhami et al. (2010) offered the adaptive learning rate and the fuzzy balancer into FSL, where the learning rate and the temperature factor would change with current state. Wiering and Hasselt (2007) combined with action probability of five different algorithms (Q-learning, Sarsa algorithm, AC, QV-learning, ACLA) in a single agent, which was an ensemble algorithm to choose an action. Hence, their researches specified the parameters (learning rate, discount rate, and temperature factor) and the probability of action selection would visibly affect the results of Q-learning or Sarsa algorithm. This paper neither studies how to select an action strategy nor studies how to adjust the parameters of RL, instead, we consider the other challenge that how to combine Q-learning and Sarsa algorithm to multiple their advantages. We will present a simple method that is called "backward Q-learning," which does not directly affect the probability of a selected action. The backward Q-learning not only can integrate with other RL algorithm, but also can enable converge more quickly to true Q-values and improve final performance.

The organization of this paper is as follows. Section 2 describes the Q-learning, the Sarsa algorithm, and a number of speed-up RL methods which will be utilized in the experiments. Section 3 explains in detail the framework of the proposed backward Q-learning by combining the other RL algorithms. In Section 4, the results of three simulations demonstrate that the presented method is useful. Section 5 discusses the results of these three simulations. Finally, the conclusions of this paper are drawn in Section 6.

2. Background and previous methods for reinforcement learning

The standard architecture of the RL algorithm is given in Fig. 1. The agent and environment interact continually, the policy selects an action a_t in current state s_t, and then the environment will respond to the action a_t and present a new situation s_{t+1}.

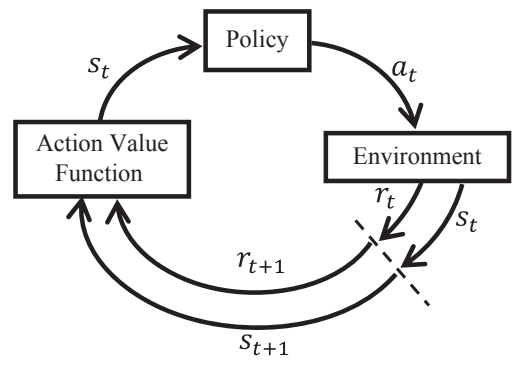

Fig. 1. The standard architecture of the RL algorithm.

The environment will also immediately supply a reward r_t. In this section, we briefly describe the Q-learning, the Sarsa algorithm, and a number of improved RL algorithms that will be used in our experiments.

2.1. Q-learning algorithm

Q-learning is one of the most important RL algorithms, which was proposed by Watkins and Dayan (1992). Its simplest form is one-step Q-learning, which is defined by

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)) \quad (1)$$

where α is the learning rate, γ is the discount factor, and r_{t+1} is the immediate reward received from the environment by taking a_t in s_t at moment t. $Q(s_t,a_t)$ estimates the action value after applying action a_t in state s_t. The probability of selecting action a_t from current state s_t is adopted by the Boltzmann distribution in the exploration strategy:

$$\Pr(a_i) = \frac{\exp(Q(s_t,a_i)/T)}{\sum_{k=1}^{n} \exp(Q(s_t,a_k)/T)} \quad (2)$$

where T is the temperature factor. Q-learning is given in Algorithm 1.

Algorithm 1. Q-learning algorithm

Initialize arbitrarily all $Q(s,a)$ values
Repeat (for each episode):
  Choose a random state s_t or initialize state s_t
  Repeat (for each step in the episode)
    Choose an action a_t from state s_t using a policy derived from Q (use Boltzmann Distribution)
    Execute the select action a_t to the environment, then receive immediate reward r_{t+1} and observe the new state s_{t+1}
    Update $Q(s_t,a_t)$ according to Eq. (1)
    $s_t \leftarrow s_{t+1}$
  Until s_t is terminal
  Recalculate the temperature parameter using the temperature-dropping criterion
Until the desired number of episodes has been searched

2.2. Sarsa algorithm

The Sarsa algorithm was proposed in Sutton and Barto (1998) and Sutton (1996), which was an improved Q-learning algorithm. The Sarsa method estimates the value of $Q(s_t,a_t)$ by applying a_t in state s_t according to the following updated formula:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \quad (3)$$

This update is done after every transition from a nonterminal state s_t. $Q(s_{t+1},a_{t+1})$ is defined as zero if s_{t+1} is terminal. Every element of the quintuple events (s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}) is used in the above updated formula, which is a transition from one state-action pair to the next. Thus, this quintuple gives rise to the name, Sarsa, for this algorithm (Sutton and Barto, 1998). The Sarsa algorithm is described in Algorithm 2.

Algorithm 2. Sarsa algorithm

Initialize arbitrarily all $Q(s,a)$ values
Repeat (for each episode):
  Choose a random state s_t or initialize state s_t
  Choose an action a_t from state s_t using the policy derived from Q (use Boltzmann Distribution)
  Repeat (for each step in the episode)
    Execute the select action a_t to the environment, then receive immediate reward r_{t+1} and observe the new state s_{t+1}
    Choose an action a_{t+1} from state s_{t+1} using policy derived from Q (use Boltzmann Distribution)
    Update $Q(s_t,a_t)$ according to Eq. (3)
    $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$
  Until s_t is terminal
  Recalculate the temperature parameter using the temperature-dropping criterion
Until the desired number of episodes has been searched

2.3. Adaptive Q-learning

In Hwang et al. (2004), an adaptive Q-learning method was presented, which was a directed approach to adjust learning rate α, discount rate γ, and temperature factor T in Boltzmann distribution. This method enables the acceleration of the convergence rate and avoids convergence to a local optimum. An adaptive Q-learning algorithm is given in Algorithm 3, where $\delta = (r_{t+1} + \gamma_t \max_a Q(s_{t+1},a) - Q(s_t,a_t))$.

Algorithm 3. Adaptive Q-learning algorithm

Initialize $Q(s,a)$ arbitrarily, and set TH_P, TH_N, C, β, and k
Repeat (for each episode):
  Choose a random state s_t or initialize state s_t
  Repeat (for each step of the episode)
    Choose an action a_t from state s_t using the policy derived from Q (use Boltzmann Distribution)
    Take action a_t, observe r_{t+1},
    IF $TH_N - kt < \max_a Q(s_{t+1},a) < TH_P + kt$
    THEN $\gamma_t(\delta,\gamma_{t-1}) = \gamma_{initial}$ and
      $\alpha_t(\delta,t) = \alpha_{initial}$
    ELSE
      $\gamma_t(t,\gamma_{t-1}) = tanh(t\gamma_{t-1})$
      $\delta = r_{t+1} + \gamma_t(t,\gamma_{t-1}) \max_a Q(s_{t+1},a) - Q(s_t,a_t)$
      $\alpha_t(\delta,t) = \tanh(\beta|\delta|/t)$
      ($\beta,k \in R_+$ and β, $k \geq 1$ and t is the number of trials.)
    $T(t) = C/t$, $C \in N$
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t(\delta,t)\delta$
    $s_t \leftarrow s_{t+1}$
  Until s_t is terminal
Until the desired number of episodes has been searched

2.4. SA-Q-learning

In order to balance exploration and exploitation of Q-learning, the modified Q-learning algorithm based on the Metropolis criterion, an SA-Q-learning algorithm was presented in Guo et al. (2004). Hence, the SA-Q-learning is given in Algorithm 4.

Algorithm 4. SA-Q-learning algorithm

Initialize arbitrarily all $Q(s,a)$ values
Repeat (for each episode):
  Choose a random state s_t or initialize state s_t
  Repeat (for each step in the episode)
    Select an action a_r in $A(s)$ arbitrarily
    Select an action a_p in $A(s)$ according to the policy
    Generate random value $\xi \in (0,1)$
    IF $\xi < \exp((Q(s,a_r) - Q(s,a_p))/T)$

Q basique, déjà implémenté

simulated annealing (réduire expl rate)

THEN $a \leftarrow a_r$
ELSE $a \leftarrow a_p$
Execute the action a, receive an immediate reward r_{t+1}, then observe the new state s_{t+1}
Update $Q(s_t,a_t)$ according to Eq. (1)
$s_t \leftarrow s_{t+1}$
Until s_t is one of the goal states
Recalculate the temperature parameter using the temperature-dropping criterion
Until the desired number of episodes has been searched

ESL = enhanced Sarsa, EQL = enhanced Q

2.5. Adaptive learning rate and fuzzy balancer

An enhanced fuzzy Sarsa learning (EFSL) presented in Derhami et al. (2010) applied the adaptive learning rate and the fuzzy balancer into FSL (Derhami et al., 2008). The adaptive learning rate is to prevent the overfitting problem in approximating the action value function, and the fuzzy balancer is to manage the balance between exploration and exploitation by generating a suitable temperature factor for the Softmax formula.

In this paper, we use Q-learning or the Sarsa algorithm to combine with the adaptive learning rate and the fuzzy balancer, which are called enhanced Q-learning (EQL) and enhanced Sarsa learning (ESL). Hence, we redefine the adaptive learning rate α_t as

$$\alpha_t = \min(c/VN(s_t), \alpha_{max}) \tag{4}$$

where c is a constant, $VN(s_t)$ is visiting number of state s_t, and α_{max} is the upper bound of α. Also, the three inputs for the fuzzy balancer are defined below:

1. The weighted difference of maximum and minimum action values in the current state s_t is redefined as

$$\tilde{Q}_t = \max_a(Q(s_t,a)) - \min_a(Q(s_t,a)) \tag{5}$$

2. The difference value of the current state s_t and the former state s_{t-1} is redefined as

$$\Delta V_t = \max_a(Q(s_t,a)) - \max_a(Q(s_{t-1},a)) \tag{6}$$

3. The exploration degree in a recent step is

$$E_t = vE_{t-1} + (1-v)\mathrm{Log}(T_{t-1}), \tag{7}$$

where $0 < v < 1$ is a weighting factor.

Then, the rule-based for the fuzzy balancer is defined below (Derhami et al., 2010):

1. If $\tilde{Q}$ is *Low* and ΔV is *Neg*, then Log(T)=*Low*.
2. If $\tilde{Q}$ is *Low* and ΔV is *Pos* and E is *Low*, then Log(*T*)=*VLow*.
3. If $\tilde{Q}$ is *Low* and ΔV is *Pos* and E is *High*, then Log(*T*)=*Low*.
4. If $\tilde{Q}$ is *High* and ΔV is *Neg*, then Log(T)=*Low*.
5. If $\tilde{Q}$ is *High* and ΔV is *Pos*, then Log(*T*)=*Low*.

The fuzzy output is a weighted average of the consequences of the rule, which can be written as

$$\mathrm{Log}(T) = \sum_{i=1}^{5} \mu_i^B O_i \tag{8}$$

where μ_i^B is the normalized firing strength, and O_i is the consequence of i-th rule of fuzzy balancer. Therefore, the EQL algorithm is shown as follows.

Algorithm 5. EQL algorithm

Initialize arbitrarily all $Q(s,a)$ values
Repeat (for each episode):
Choose a random state s_t or initialize state s_t
Repeat (for each step in the episode)
Compute $\tilde{Q}_t$, ΔV_t, *and* E_t using Eqs. (5)–(7), respectively.
Compute T using the fuzzy balancer
Using T to choose a suitable action a_t from state s_t using the policy derived from Q (use Boltzmann Distribution)
Execute the select action a_t to the environment, then receive immediate reward r_{t+1} and observe the new state s_{t+1}
Compute the adaptive learning rate α_t using Eq. (4)
Update $Q(s_t,a_t)$ according to Eq. (1)
$s_t \leftarrow s_{t+1}$
Until s_t is terminal
Until the desired number of episodes has been searched

3. Backward Q-learning

Q-learning and Sarsa algorithm have been applied to many fields such as robotics, artificial intelligence, and mechatronics control. Nevertheless, there are still some problems with Q-learning and the Sarsa algorithm. The problems consist of how to accelerate the convergence rate, balance the exploration and exploitation dilemma, and avoid converging to a local minimum (Derhami et al., 2010; Guo et al., 2004; Hwang et al., 2004; Lin and Li, 2003; Sutton and Barto, 1998; Wiering and Hasselt, 2007, 2008). Hence, in order to solve these problems, the majority of speed-up RL algorithms have been studied in relation to action selection strategy in recent years (Derhami et al., 2010; Guo et al., 2004; Hwang et al., 2004; Wiering and Hasselt, 2007, 2008). In addition, Sutton and Barto (1998) performed Q-learning and the Sarsa algorithm in cliff-walk problem, and the experimental results had been shown that the Sarsa algorithm has a faster convergence rate than Q-learning, while Q-learning possesses the better final performance. Thus, we consider how to combine the Q-learning and the Sarsa algorithm in order to achieve a faster convergence rate and a better final performance.

Based on these considerations, we present a simple method called "backward Q-learning," which is different from the other speed-up RL algorithms (Derhami et al., 2010; Guo et al., 2004; Hwang et al., 2004; Lin and Li, 2003; Wiering and Hasselt, 2007, 2008). The architecture of the RL algorithm integrated with backward Q-learning is given in Fig. 2. As the figure shows, backward Q-learning is a directly affected action value function and then an

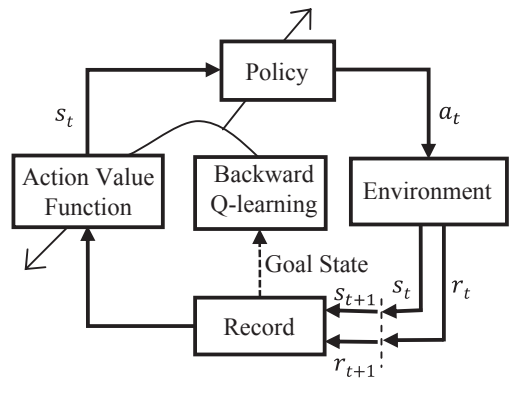

Fig. 2. The architecture of the RL algorithm integrated with backward Q-learning.

indirectly affected action selection policy. Hence, it can be easily implemented in the Sarsa algorithm or in another RL algorithm. As the agent interacting with the environment increases, accurate knowledge of the agent also increases. We considered how the agent can enhance the learning speed, balance the exploration–exploitation dilemma, and converge to the global minimum by using these previous states, actions, and information in one episode. Therefore, we record that these went through states, elected actions, and acquired rewards in an episode, and then this information will be utilized to update the Q-learning function again. It is not an ordinary Q-learning updating function and is backward updated, so this method is called "backward Q-learning."

When the agent achieves the goal state in the current episode, we will utilize the data that had occurred to backward update the Q-learning function. For instance, the state s_{t-n} is defined as a starting state, and the state s_{t+1} is defined as a terminal state. The agent updates the Q-learning function N times from starting state s_{t-N} to terminal state s_{t+1} in one episode. And the agent simultaneously records the every element of the four events (s, a, r, s'), then the agent will utilize the information to backward update the Q-learning function N times. Therefore, we redefine the one-step Q-learning function as

$$Q(s_t^i, a_t^i) \leftarrow Q(s_t^i, a_t^i) + \alpha(r_{t+1}^i + \gamma \max_a Q(s_{t+1}^i, a) - Q(s_t^i, a_t^i)) \tag{9}$$

for $i=1,2,\ldots,N$, where i is the number of times for updated the Q-learning function in current episode. And the agent simultaneously records the four events in M^i, that is

$$M^i \leftarrow s_t^i,\ a_t^i,\ r_{t+1}^i,\ s_{t+1}^i. \tag{10}$$

when the agent reaches the goal state, the agent will backward update the Q-learning function based on the information of M^i as follows:

$$Q(s_t^j, a_t^j) \leftarrow Q(s_t^j, a_t^j) + \alpha_b(r_{t+1}^j + \gamma_b \max_a Q(s_{t+1}^j, a) - Q(s_t^j, a_t^j)) \tag{11}$$

where $j=N,N-1,N-2,\ldots,1$. Hence, the agent can utilize the previous information to backward update the Q-learning function. The backward Q-learning based Sarsa algorithm (BQSA) is presented in Algorithm 6, where α_b and γ_b are the learning rate and discount factor for the backward Q-learning.

Algorithm 6. BQSA algorithm

Initialize arbitrarily all $Q(s,a)$, M and set α_b and γ_b
Repeat (for each episode):
  Choose a random state s_t or initialize state s_t
  Choose an action a_t from state s_t using the policy derived from Q (use Boltzmann Distribution)
  Repeat N times (for each step in the episode)
    Execute the select action a_t^i to the environment, then receive immediate reward r_{t+1}^i and observe the new state s_{t+1}^i
    Choose an action a_{t+1}^i from state s_{t+1}^i using the policy derived from Q (use Boltzmann Distribution)
    Record the four events in $M^i \leftarrow s_t^i,\ a_t^i,\ r_{t+1}^i,\ s_{t+1}^i$.
    Update $Q(s_t^i, a_t^i)$ according to Eq. (9)
    $s_t^i \leftarrow s_{t+1}^i$; $a_t^i \leftarrow a_{t+1}^i$
    $i \leftarrow i+1$
  Until s_t is terminal state
  For $j=N$ to 1
    Backward update $Q(s_t^j, a_t^j)$ according to Eq. (11).
  End for
  Initialize all M values
  Recalculate the temperature parameter using the temperature-dropping criterion
Until the desired number of episodes has been searched

Fig. 3 shows the Sarsa algorithm and BQSA method. One can clearly find that BQSA records four events in $M^i \leftarrow s_t^i,\ a_t^i,\ r_{t+1}^i,\ s_{t+1}^i$ immediately, and uses M^i information to backward update $Q(s_t^j, a_t^j)$ according to Eq. (11) when the agent reaches the goal state.

4. Simulations

In order to assess the learning performance of using backward Q-learning to integrate with the other RL algorithm, we will compare them to the other RL methods introduced in Section 2. Hence, we perform three experiments with the cliff-walking, the mountain car problem and the cart–pole balancing control system to demonstrate our method. The first two experiments are the

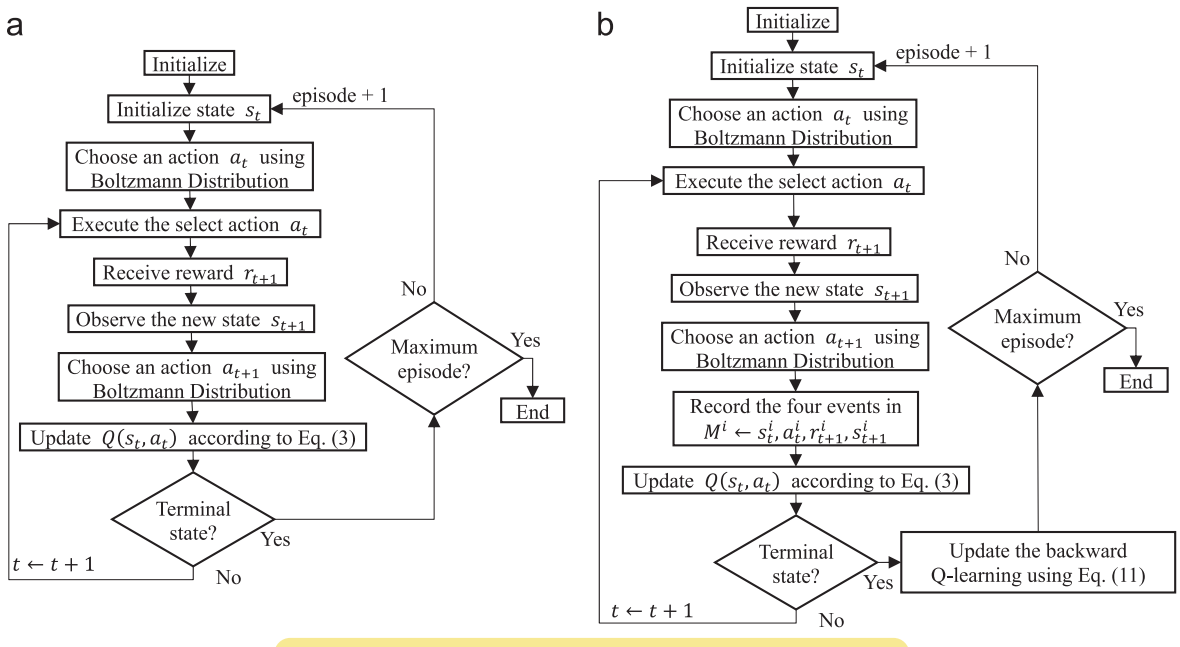

Fig. 3. (a) The flow chart of Sarsa algorithm. (b) The flow chart of BQSA.

benchmark for the RL algorithm (Bianchi et al., 2008; Guo et al., 2004; Sutton and Barto, 1998; Wiering and Hasselt, 2007, 2008); the third experiment, cart–pole balancing control system, is in fact a benchmark problem for the designs of nonlinear, fuzzy, or intelligent controllers (Belarbi et al., 2005; Bratko, 1997; El-Hawwary et al., 2006; Oh et al., 2004; Gurumoorthy and Sanders, 1993; Li and Shieh, 2000). One can easily find from the nonlinear dynamical equation of the the cart–pole balancing control system that it is a nonlinear and unstable plant. Moreover, suppose both the pole angle and cart position are considered the difficulty about non-minimum phase phenomenon reveals (Gurumoorthy and Sanders, 1993; Li and Shieh, 2000). The cart–pole balancing control system addressed in Juang (2005) is a more complex environment for the RL algorithm.

Example 1. Cliff-walking: The cliff-walking shown in Fig. 4 has been used as a benchmark problem for the evaluation of Q-learning and the Sarsa algorithm (Sutton and Barto, 1998) for decades. This cliff-walking consists of 12×5 states, and there are four actions $A = \{\text{north, east, south, west}\}$ in each state except for the region marked "The Cliff." The purpose of the agent is to learn how to arrive at goal state G from the random starting state S except for the region of "The Cliff."

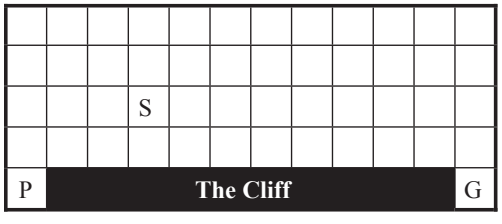

Fig. 4. The 12×5 maze of cliff-walking. The random starting state is indicated by S for learning phase. The starting state P is one of the starting position for testing phase. And the goal state is indicated by G.

Experiment setup: The reward is 10 when the agent arrives at goal state G, the reward is -10 when the agent steps into the region marked "The Cliff," and then the agent will be instantly sent back to the starting state S. For other steps, the agent receives reward -1. The experiment has average results of 100 runs. A run is composed of a "learning phase" and a "testing phase." The learning phase ends when the agent reaches 300 times to the goal state G. The testing phase ends when the agent arrives at the goal from every starting state except the region of "The Cliff" and goal state G. And the testing episode ends when the agent reaches the goal G or the number of steps exceeds 40. Therefore, the testing phase is made up of 49 episodes. For the learning phase, the number of time-steps to arrive at the goal state is called the Length of Episode (LE), which is a measure of time-steps to reach the goal state (Derhami et al., 2010). And we call the total number of time-steps from every starting state to the goal state in the testing phase as the Total Steps (TS), which is a measure of the action quality.

We use the tabular specification and first perform the experiments to find the best parameters possibly for different RL algorithms. Table 1 shows the average experimental results, cumulative rewards, learning time, and total performance of 100 runs of experiments for every RL algorithm introduced in Section 2. The results show that ESL is the highest cumulative reward, the least LE and learning time, which also show that ESL converges more quickly for the learning phase. On the other hand, the EQL is the fewest TS and a higher success rate, which means that EQL can find the approximate optimal path planning in each experiment. Also, the Sarsa algorithm is higher cumulative rewards, and fewer LE and standard deviation than Q-learning for the learning phase; the Q-learning has fewer TS than the Sarsa algorithm for the testing phase. This means the Sarsa algorithm has the characteristics of convergence faster, while the Q-learning has a better final performance.

In order to choose the parameters of backward Q-learning, we select 5×5 groups ($\alpha_b = 0.1, 0.3, 0.5, 0.7, 0.9$, $\gamma_b = 0.15, 0.35, 0.55, 0.75, 0.95$) to perform and average 100 runs of each group for the

Table 1
Result of average 100 runs for each RL algorithm in Example 1.

Methods	LE	Std.	Cumulative	TS	Success	Time	α	γ
Q-learning	35.767	10.868	−35.492	405.22	1	1.238	0.9	0.95
Sarsa algorithm	31.029	6.826	−26.491	427.74	1	1.020	0.5	0.95
Adaptive Q-learning	34.448	0.924	−26.782	422.23	0.978	1.271	0.9	0.95
EQL	27.549	1.800	−18.310	397.56	1	1.420	ada.	0.95
ESL	13.765	0.413	−3.601	405.97	1	0.732	ada.	0.95
SA-Q-learning	25.612	4.819	−19.854	405.72	1	0.849	0.5	0.95

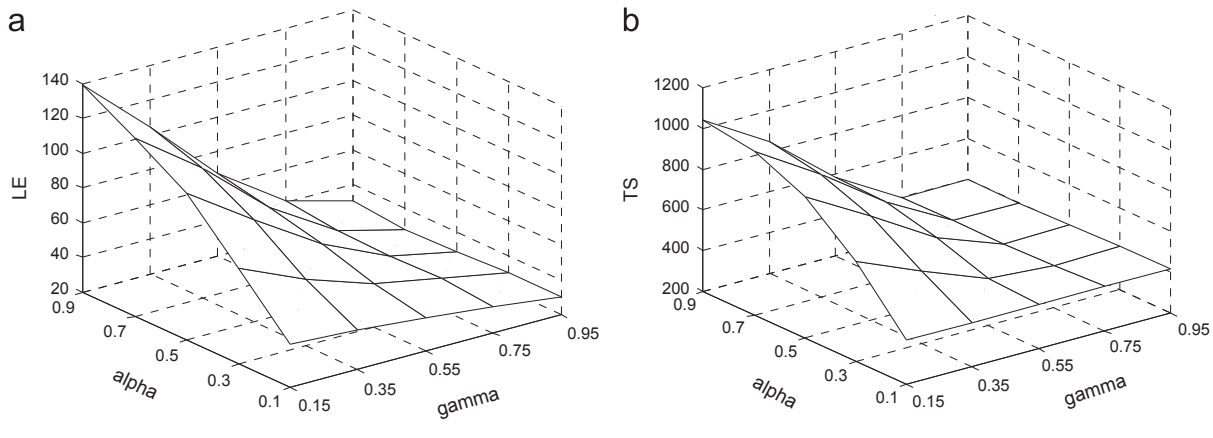

Fig. 5. (a) Results of LE in different parameters for BQSA. (b) Results of TS in different parameters for BQSA.

Table 2
Result of average 100 runs for each RL algorithm combined with backward Q-learning in Example 1.

Methods	LE	Std.	Cumulative	TS	Success	Time	α	γ	α_b	γ_b
Q-learning	31.096	10.412	−28.567	397.94	1	1.297	0.9	0.95	0.5	0.95
Sarsa algorithm	27.802	5.891	−22.794	404.56	1	1.117	0.5	0.95	0.5	0.95
Adaptive Q-learning	33.166	0.741	−25.500	397	1	1.487	0.9	0.95	0.5	0.95
EQL	25.741	1.755	−16.406	397.02	1	1.452	ada.	0.95	0.5	0.95
ESL	12.365	0.345	−2.185	400.86	1	0.730	ada.	0.95	0.5	0.95
SA-Q-learning	23.144	4.885	−16.216	402.60	1	0.961	0.5	0.95	0.5	0.95

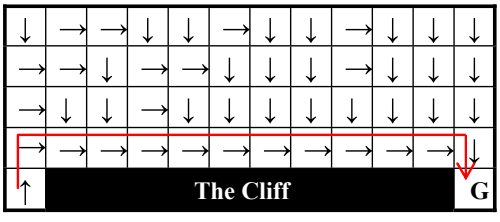

Fig. 6. Q-learning implemented in cliff-walking.

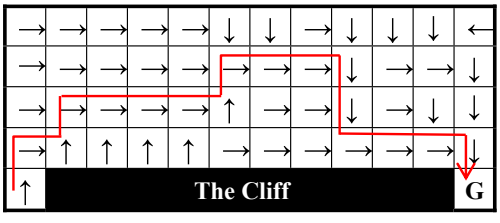

Fig. 7. Sarsa algorithm implemented in cliff-walking.

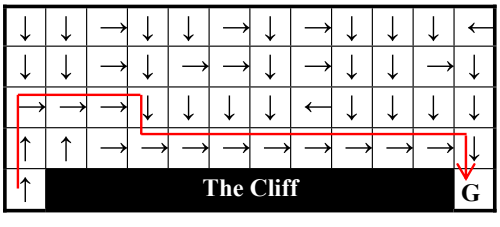

Fig. 8. ESL implemented in cliff-walking.

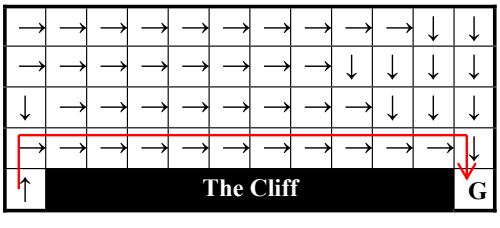

Fig. 9. BQQ implemented in cliff-walking.

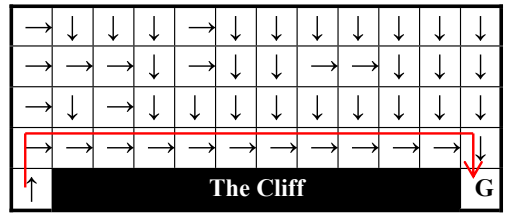

Fig. 10. BQSA implemented in cliff-walking.

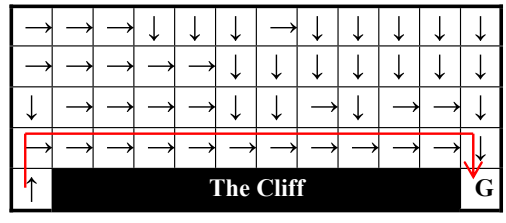

Fig. 11. BQESL implemented in cliff-walking.

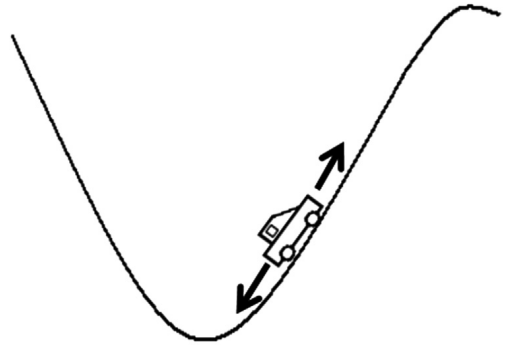

Fig. 12. Mountain car.

BQSA method as shown in Fig. 5. Hence, we can clearly find that the performance and TS are better when α_b is smaller and γ_b is bigger. According to this simulation, we select $\alpha_b=0.5$, $\gamma_b=0.95$.

In order to make a fair comparison among the different RL algorithms, we set the same parameter of backward Q-learning $\alpha_b=0.5$, $\gamma_b=0.95$ to all different RL methods in this example. Table 2 illustrates the experimental results of the RL algorithms integrated with backward Q-learning. The experimental setup and parameters of the RL algorithms are the same as those in Table 1. The experimental results show that the backward Q-learning not only causes LE decreases and cumulative rewards increases in learning phase, but also leads to a TS decreases for the testing phase. Notice that applying backward Q-learning to Q-learning, adaptive Q-learning or EQL will achieve that the resulting path planning is very closer to the shortest path. Although the backward Q-learning spends more computation time, the backward Q-learning based RL algorithms can decrease the number of learning and find the near optimal path planning.

Figs. 6–8, respectively, depict the action quality of Q-learning, the Sarsa and ESL algorithm for the testing phase, where Q-learning can find a better path planning than the Sarsa algorithm. Basing on the average results of 100 runs, the Sarsa algorithm ($\alpha=0.5,\gamma=0.95$) needs 15.8 steps to achieve the goal state G from the starting state P, the ESL ($\alpha=$ada.,$\gamma=0.95$) spends 13.8 steps, the Q-learning ($\alpha=0.9,\gamma=0.95$) consumes 13.74 steps. It clearly shows that Q-learning offers a closer optimum path planning from starting state P to goal state G. RL algorithms combined with backward Q-learning are shown in Figs. 9–11, respectively, where the BQSA ($\alpha=0.5,\gamma=0.95,\alpha_b=0.9,\gamma_b=0.95$) only needs 13.02 steps, the backward Q-learning based ESL (BQESL) ($\alpha=$ada.,$\gamma=0.95$, $\alpha_b=0.3,\gamma_b=0.95$) only spends 13.4 steps and the backward Q-learning based Q-learning (BQQ) ($\alpha=0.9,\gamma=0.95,\alpha_b=0.9,\gamma_b=0.95$) just only consumes 13 steps, which shows that BQQ can always learn the optimum path. The experimental results clearly describe that backward Q-learning is useful to learn a closer optimal action. Therefore, the presented approach integrates with Q-learning, the

Sarsa algorithm, or other speed-up RL algorithms are valid for improving LE, action quality, and total performance.

Example 2. Mountain car: In this problem, the goal is to drive a car to the top of a hill as shown in Fig. 12. Because gravity provides a greater force than a car, the car cannot easily speed up to reach the goal, even at its full throttle. The driver has to push back and forth to obtain sufficient velocity to climb the hill. The agent of RL has to make a correct decision so as to choose an action that will arrive at the goal. The input variables are car position x and car velocity v. The system dynamics of the car are described by (Derhami et al., 2010)

$$\begin{cases} v_{t+1} = \min(0.07,\ \max(-0.07,\ v_t + 0.001 \times F_t - 0.0025 \times \cos(3x_t))) \\ x_{t+1} = \min(0.5,\ \max(-1.2,\ x_t + v_{t+1})) \end{cases} \tag{12}$$

where $x \in \{-1.2, 0.5\}$, $v \in \{-0.07, 0.07\}$, and $F \in \{-1, 0, 1\}$. When $x_t = -1.2$, the car velocity is set to zero. The goal arrives at the position $x_t = 0.5$.

Experiment setup: The input variables (x, v) are quantized into $11 \times 11 = 121$ states. The reward is 200 when the car reaches the goal state, and the reward is -1 for the other step. An episode ends when the agent arrives at the position $x_t = 0.5$ or the number of steps exceeds 2500. For the learning phase and testing phase, we generated 300 and 40 pairs of random values for the initial car position and velocity. The performance of this experiment is the average of 100 runs, and all experiments have the same set of random starting states.

In order to choose the parameters of backward Q-learning, 5×5 groups ($\alpha_b = 0.01, 0.03, 0.05, 0.07, 0.09$, $\gamma_b = 0.19, 0.39, 0.59, 0.79, 0.99$) are selected to perform and average 100 runs of each group for the BQSA method as shown in Fig. 13. One can readily ascertain that the performance and successful rate are better when α_b is smaller and γ_b is bigger. From this simulation result, we select $\alpha_b = 0.05$, $\gamma_b = 0.99$.

Table 3 lists the average experimental results, cumulative rewards, learning time, and total performance of 100 runs for each RL algorithm that introduced in Section 2. One can read from Table 3 that ESL requires the least number of LE to reach the goal state. And ESL has the highest success rate than other RL algorithms and demonstrates the adaptive learning rate and the fuzzy balancer is useful to enhance learning speed and a better performance.

On the other hand, the Sarsa algorithm gives fewer LE and higher cumulative rewards than Q-learning for the learning phase. For comparisons, the simulation results of utilizing the presented method to combine with RL algorithms are given in Table 4. The experimental setup and parameters of the RL algorithms are the same as those in Table 3. The proposed method based RL algorithms can enhance learning speed, achieve better final performance, and lead to higher success rate for the mountain car problem. Notice that if the agent exceeds the number of 2500

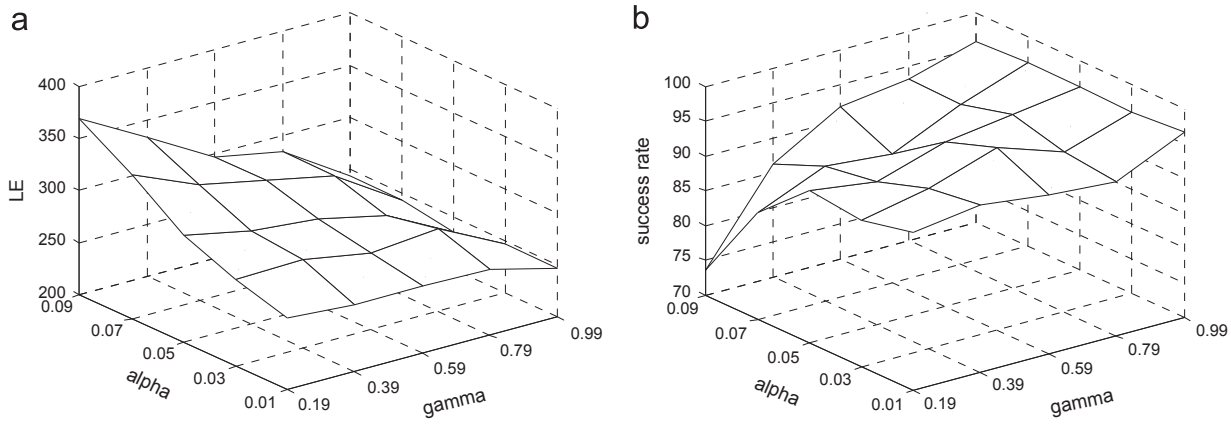

Fig. 13. (a) Results of LE in different parameters for BQSA. (b) Results of success rate in different parameters for BQSA.

Table 3
Result of 100 runs for each RL algorithm in Example 2.

LE = Length of episode

Methods	LE	Std.	Cumulative	TLE	Success	Time	α	γ
Q-learning	298.405	56.638	−108.614	118.804	0.940	7.896	0.5	0.99
Sarsa algorithm	255.612	54.600	−62.746	119.384	0.937	6.133	0.5	0.99
Adaptive Q-learning	197.359	35.276	1.202	126.706	0.861	5.408	0.5	0.99
EQL	158.093	81.426	40.442	97.068	0.968	6.388	ada.	0.99
ESL	140.908	33.479	58.933	102.125	0.972	5.344	ada.	0.99
SA-Q-learning	157.765	30.010	41.165	116.138	0.943	3.889	0.7	0.99

Table 4
Result of 100 runs for each RL algorithm combined with backward Q-learning in Example 2.

Methods	LE	Std.	Cumulative	TLE.	Success	Time	α	γ	α_b	γ_b
Q-learning	267.300	37.850	−75.975	112.567	0.983	8.272	0.5	0.99	0.05	0.99
Sarsa algorithm	239.272	47.329	−45.876	112.575	0.969	6.949	0.5	0.99	0.05	0.99
Adaptive Q-learning	192.604	33.462	6.500	121.156	0.898	6.123	0.5	0.99	0.05	0.99
EQL	144.703	45.925	54.428	98.406	0.983	6.724	ada.	0.99	0.05	0.99
ESL	132.844	30.283	67.178	98.295	0.982	5.852	ada.	0.99	0.05	0.99
SA-Q-learning	154.565	32.195	44.251	107.320	0.978	4.246	0.7	0.99	0.05	0.99

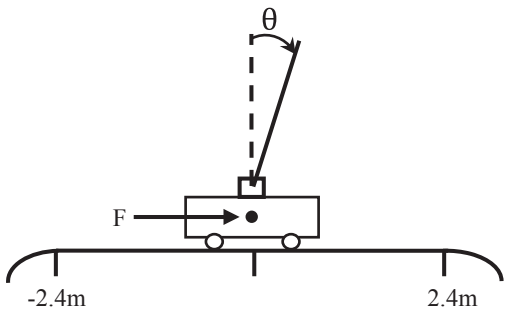

Fig. 14. Cart–pole balancing system.

steps in an episode, the agent will not utilize the backward Q-learning, because it is a failed exploration.

Example 3. Cart–pole balancing control system: In order to verify the performance of the presented method under a nonstationary problem, a cart–pole balancing control system is adopted as shown in Fig. 14. The dynamics of the cart–pole balancing control system addressed in Juang (2005) are modeled as

$$\theta(t+1)=\theta(t)+\Delta\dot{\theta}(t) \tag{13}$$

$$\dot{\theta}(t+1)=\dot{\theta}(t)+\Delta\frac{mg\ \sin\ \theta_t-\cos\ \theta_t[F+m_p\ell\dot{\theta}_t^2\ \sin\ \theta_t]}{(\frac{4}{3})m\ell-m_p\ell\ \cos^2\theta_t} \tag{14}$$

$$x(t+1)=x(t)+\Delta\dot{x}(t) \tag{15}$$

$$\dot{x}(t+1)=\dot{x}(t)+\Delta\frac{F+m_p\ell[\dot{\theta}_t^2 sin\theta_t-\ddot{\theta}_t cos\theta_t]}{m} \tag{16}$$

where θ is the angle of the pole from an upright position (in degree), $\dot{\theta}$ presents the angular velocity of the pole (in degree/second), x is the position of the center of the cart (in meters), and $\dot{x}$ denotes the velocity of the cart (in meters/second). The cart–pole balancing control system has been a benchmark problem for the designs of nonlinear, fuzzy, or intelligent controllers for decades (Belarbi et al., 2005; Bratko, 1997; El-Hawwary et al., 2006; Gurumoorthy and Sanders, 1993; Li and Shieh, 2000; Oh et al., 2004). One can easily find from the nonlinear dynamical equations (13)–(16) that it is a nonlinear and unstable plant. Moreover, suppose both the pole angle and cart position are considered the difficulty about the non-minimum phase phenomenon reveals (Gurumoorthy and Sanders, 1993; Li and Shieh, 2000). The control action is $F\in[-10,\ -5,\ 0,\ 5,\ 10]$, which is the force applied to the cart to move right or left, is the combined mass of the pole and the cart, m_p=0.1 kg is mass of the pole, g=9.8 m/s^2 denotes the gravity acceleration, $\ell=0.5\ m$ is the length of the pole, and the sampling interval $\Delta=0.02$ (s).

Experiment setup: The goal of the agent is to determine the control force applied to the cart to balance the pole upright. The four input variables $(\theta,\dot{\theta},x,\dot{x})$ are quantized into $6\times3\times3\times3=162$ boxes according to the quantization method in Barto et al. (1983). The reward is -10 when the control fails, and the reward is 0 for the other step. A failure trial defines the pole angle ($|\theta|\le12^\circ$) and the cart position ($|x|\le2.4$ m). A run comprises 5000 trials and it is successful when the pole is upright for 120 000 time steps. The initial values of $(\dot{\theta},\dot{x})$ are all set to zero, $|x|\le0.5$ and $|\theta|\le4^\circ$ are randomly generated for each trial. The performance of this experiment is the average of 50 runs. A failure run is defined as no successful trial to balance the pole after 5000 trials.

The performance comparisons of Q-learning, Sarsa algorithm, and SA-Q-learning based cart–pole balancing control system are shown in Table 5. The experimental results show that the SA-Q-learning needs the least number of trials and learning time. And the performance comparisons of average 50 runs for RL algorithms based on backward Q-learning are tabulated in Table 6, where the SA-Q-learning based on backward Q-learning needs the least number of trials and learning time. One can clearly find that the proposed learning algorithm requires the less number of trials and learning time than those of original RL algorithm to upright the pole.

Table 5
Result of 50 runs for each RL algorithm in Example 3.

Methods	Trial	Std.	Time	α	γ
Q-learning	649.88	151.88	48.83	0.1	0.99
Sarsa algorithm	558.92	198.84	40.21	0.1	0.99
SA-Q-learning	559.80	189.16	33.09	0.1	0.99

Table 6
Result of 50 runs for each RL algorithm combined with backward Q-learning in Example 3.

Methods	Trial	Std.	Time	α	γ	α_b	γ_b
Q-learning	553.18	146.52	46.46	0.1	0.99	0.01	0.99
Sarsa algorithm	461.74	136.31	39.68	0.1	0.99	0.01	0.99
SA-Q-learning	481.92	135.20	33.48	0.1	0.99	0.01	0.99

5. Discussion

Three experiments are presented to illustrate the feasibility of the proposed schemes. The cliff-walking is a small maze, a 60×4 Q-table is good enough. The mountain car problem has a 121×3 Q-table, while the cart–pole balancing control system has to build a larger 162×5 Q-table. RL usually needs more learning time, and increases the probability to stick in the local minimum if the RL has a larger Q-table. Hence, three experiments have different size of Q-tables to demonstrate and compare the presented backward Q-learning with other methods. The proposed method decreases the number of times of LE and trials from the simulation results, it means that the agent can decrease the number of times in executing an action. Besides, for practical engineering applications, Rahimiyan and Mashhadi (2010) applied Q-learning to power supplier, and Kamio and Iba (2005) executed genetic programming and Q-learning to enable a real robot to adapt its actions in a real environment. The learning time and the time of performing actions can be further decreased suppose that their methods are combined with our backward Q-learning. However, the only assumption and disadvantage of our method is the expense in an extra memory for storing the occurred events in one episode. With the development of semiconductors and computers, we believe that memory size is really not a difficult issue.

In general, RL methods based on the backward Q-learning consume more computational time. However, a learning method with a smaller number of trials is usually able to imply a shorter experimental time in practical applications, because the actions taken in the real-world applications require much more time than the actual learning computational time (Wiering and Hasselt, 2008).

6. Conclusions

In this paper, we have presented a new method, called the backward Q-learning, which is successive to combine with the Sarsa algorithm and the Q-learning. For the simulation of cliff-walk problem, it can find that Sarsa algorithm converges

faster than Q-learning, but Sarsa algorithm cannot find the optimum path planning. Thus, using the backward Q-learning based Sarsa algorithm cannot only enhance learning speed but also improve the final performance. Besides, the backward Q-learning also can easily combine with the Q-learning or other RL algorithms. All the simulations demonstrate that the backward Q-learning based RL algorithms can enhance learning speed and improve action quality and total performance. The memory size is the limitation of the proposed scheme because the backward Q-learning needs to record four events in every step. Both decreasing the memory size and computational time and applying the presented methods to some practical engineering applications are worthy of future studies.

Acknowledgments

This research was, in part, supported by Ministry of Education, Taiwan, ROC, The aim for the Top University Project to the National Cheng Kung University (NCKU). This work was also supported by the National Science Council of the Republic of China under Grant NSC101-2221-E-006-193-MY3.

References

Aissani, N., Beldjilali, B., Trentesaux, D., 2009. Dynamic scheduling of maintenance tasks in the petroleum industry: a reinforcement approach. Engineering Applications of Artificial Intelligence 22, 1089–1103.
Abdi, J., Moshiri, B., Abdulhai, B., Sedigh, A.K., 2012. Forecasting of short-term traffic-flow based on improved neurofuzzy models via emotional temporal difference learning algorithm. Engineering Applications of Artificial Intelligence 25, 1022–1042.
Barto, A.G., Sutton, R.S., Anderson, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics 13 (5), 834–846.
Boyan, J.A., Moore, A.W., 1995. Generalization in reinforcement learning: safely approximating the value function. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (Eds.), Advances in Neural Information Processing Systems, vol. 7. MIT Press, Cambridge MA, pp. 369–376.
Bratko, I., 1997. Transfer of control skill by machine learning. Engineering Applications of Artificial Intelligence 10 (1), 63–71.
Belarbi, K., Titel, F., Bourebia, W., Benmahammed, K., 2005. Design of Mamdani fuzzy logic controllers with rule base minimisation using genetic algorithm. Engineering Applications of Artificial Intelligence 18 (7), 875–880.
Bianchi, R.A.C., Ribeiro, C.H.C., Costa, A.H.R., 2008. Accelerating autonomous learning by using heuristic selection of actions. Journal of Heuristics 14 (2), 135–168.
Boubertakh, H., Tadjine, M., Glorennec, P.Y., 2010. A new mobile robot navigation method using fuzzy logic and a modified Q-learning algorithm. Journal of Intelligent & Fuzzy Systems 21 (2010), 113–119.
Cheng, C., Chau, K.W., 2001. Fuzzy iteration methodology for reservoir flood control operation. Journal of the American Water Resources Association 37 (5), 1381–1388.
Chau, K.W., 2007. Application of a PSO-based neural network in analysis of outcomes of construction claims. Automation in Construction 16 (5), 642–646.
Chu, B., Hong, D., Park, J., Chung, J.H., 2008. Passive dynamic walker controller design employing an RLS-based natural actor–critic learning algorithm. Engineering Applications of Artificial Intelligence 21, 1027–1034.
Derhami, V., Majd, V.J., Ahmadabadi, M.N., 2008. Fuzzy Sarsa learning and the proof of existence of its stationary points. Asian Journal of Control 10 (5), 535–549.
Dong, D.Y., Chen, C.L., Li, H.X., Tarn, T.J., 2008. Quantum reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 38 (5), 1207–1220.
Derhami, V., Majd, V.J., Ahmadabadi, M.N., 2010. Exploration and exploitation balance management in fuzzy reinforcement learning. Fuzzy Sets Systems 161 (4), 578–595.
El-Hawwary, M.I., Elshafei, A.L., Emara, H.M., Fattah, H.A.A., 2006. Adaptive fuzzy control of the inverted pendulum problem. IEEE Transactions on Control Systems Technology 5 (2), 254–260.
Gordon, G., 1995. Stable Function Approximation in Dynamic Programming. Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-CS-95-103.
Guo, M., Liu, Y., Malec, J., 2004. A new Q-learning algorithm based on the metropolis criterion. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 34 (5), 2140–2143.
Gurumoorthy R, Sanders S. R., 1993. Controlling nonminimum phase nonlinear system-the inverted pendulum on a cart example. In: Proceedings of the American Control Conference, pp. 680–685.
Hwang, K.S., Tan, S.W., Chen, C.C., 2004. Cooperative strategy based on adaptive Q-learning for robot soccer systems. IEEE Transactions on Fuzzy Systems 12 (4), 569–576.
Harandi, M.T., Ahmadabadi, M.N., Araabi, B.N., 2009. Optimal local basis: a reinforcement learning approach for face recognition. International Journal of Computer Vision 81 (2), 191–204.
Juang, C.F., 2005. Combination of online clustering and q-value based GA for reinforcement fuzzy system design. IEEE Transactions on Fuzzy Systems 13 (3), 289–302.
Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: a survey. Journal of Artificial Intelligence Research 4 (1996), 237–287.
Kamio, S., Iba, H., 2005. Adaptation technique for integrating genetic programming and reinforcement learning for real robots. IEEE Transactions on Evolutionary Computation 9 (3), 318–333.
Li, T.H.S., Shieh, M.Y., 2000. Switching-type fuzzy sliding mode control of a cart–pole system. Mechatronics 10 (1-2), 91–109.
Lin, Y.P., Li, X.Y., 2003. Reinforcement learning based on local state feature learning and policy adjustment. Information Sciences 154 (2003), 59–70.
Li, T.H.S., Su, Y.T., Lai, S.W., Hu, J.J., 2011. Walking motion generation, synthesis, and control for biped robot by using PGRL, LPI, and fuzzy logic. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 41 (3), 736–748.
Millan, J.R., 1996. Rapid, safe, and incremental learning of navigation strategies. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 26 (3), 408–420.
Muttil, N., Chau, K.W., 2006. Neural network and genetic programming for modelling coastal algal blooms. International Journal of Environment and Pollution 28 (3–4), 223–238.
Mori, T., Ishii, S., 2011. Incremental state aggregation for value function estimation in reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 41 (5), 1407–1416.
Oh, S.K., Pedrycz, W., Rho, S.B., Ahn, T.C., 2004. Parameter estimation of fuzzy controller and its application to inverted pendulum. Engineering Applications of Artificial Intelligence 17 (1), 37–60.
Rahimiyan, M., Mashhadi, H.R., 2010. An adaptive Q-learning algorithm developed for agent-based computational modeling of electricity market. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applied Review 40 (5), 547–556.
Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. Machine Learning 3 (1), 9–44.
Sutton, R.S., 1996. Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (Eds.), Advances in Neural Information Processing Systems, 1996. MIT Press, Cambridge MA, pp. 1038–1045.
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
Sharma, R., Gopal, M., 2008. A Markov game-adaptive fuzzy controller for robot manipulators. IEEE Transactions on Fuzzy Systems 16 (1), 171–186.
Taormina, R., Chau, K.W., Sethi, R., 2012. Artificial neural network simulation of hourly groundwater levels in a coastal aquifer system of the Venice lagoon. Engineering Applications of Artificial Intelligence 25, 1670–1676.
Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. Machine Learning 8 (1992), 279–292.
Wiering, M.A., Hasselt, H. van, 2007. Two novel on-policy reinforcement learning algorithms based on TD(λ)-methods. Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, 280–287.
Wang, Y., de Silva, C.W., 2008. A machine-learning approach to multi-robot coordination. Engineering Applications of Artificial Intelligence 21, 470–484.
Wiering, M.A., Hasselt, H. van, 2008. Ensemble algorithms in reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 38 (4), 930–936.
Wu, C.L., Chau, K.W., Li, Y.S., 2009. Predicting monthly streamflow using data-driven models coupled with data-preprocessing techniques. Water Resources Research 45 (2009), W08432.
Wurm, K.M., Stachniss, C., Grisetti, G., 2010. Bridging the gap between feature- and grid-based SLAM. Robotics and Autonomous Systems 58 (2), 140–148.
Zhang, J., Chau, K.W., 2009. Multilayer ensemble pruning via novel multi-sub-swarm particle swarm optimization. Journal of Universal Computer Science 15 (4), 840–858.