Int. J. Appl. Math. Comput. Sci., 2013, Vol. 23, No. 3, 623–635
DOI: 10.2478/amcs-2013-0047

EPOCH–INCREMENTAL REINFORCEMENT LEARNING ALGORITHMS

Roman ZAJDEL

Faculty of Electrical and Computer Engineering
Rzeszów University of Technology, Al. Powstańców Warszawy 12, 35-959 Rzeszów, Poland
e-mail: rzajdel@prz.edu.pl

In this article, a new class of the epoch-incremental reinforcement learning algorithm is proposed. In the incremental mode, the fundamental TD(0) or TD(λ) algorithm is performed and an environment model is created. In the epoch mode, on the basis of the environment model, the distances of past-active states to the terminal state are computed. These distances and the reinforcement terminal state signal are used to improve the agent policy.

Keywords: reinforcement learning, epoch-incremental algorithm, grid world.

1. Introduction

In reinforcement learning algorithms, the interactions of an agent with an environment are divided into episodes or epochs. Each episode is composed of a series of agent–environment interactions. The number of these iterations is usually unknown *a priori*. Each episode ends in a special state called the terminal state (Sutton and Barto, 1998). This is a state which cannot be left.

The efficiency of the basic reinforcement learning algorithm, e.g., Q-learning (Watkins, 1989), AHC (Barto *et al.*, 1983) or Sarsa (Rummery and Niranjan, 1994), measured in the number of epochs to obtain the optimal policy is relatively small. For this reason, the number of practical implementations of these algorithms in more complex problems is rather insignificant. The unquestionable advantage of these algorithms is low computational complexity, which implies an extremely short learning time of each epoch. The short learning time is essential for application of reinforcement learning algorithms to on-line control problems. Thus, acceleration methods of reinforcement learning algorithms should ensure both relatively small computational complexity and high efficiency.

Unfortunately, the acceleration methods used up to date, which reduce the number of epochs needed to obtain the optimal policy, have required significantly more learning time. For example, one of the most often used acceleration methods, such as the temporal-differences mechanism TD($\lambda > 0$), requires additional memory elements known as eligibility traces. The learning time of TD($\lambda > 0$) grows because the updates of the policy function are made for all states, not only for one (actual), like in the class of basic reinforcement learning algorithms TD(0) (Sutton and Barto, 1998; Watkins, 1989). Learning methods such as Dyna-learning (Sutton, 1990; 1991) or prioritized sweeping (Moore and Atkeson, 1993; Peng and Williams, 1993), which are much more efficient than TD($\lambda > 0$), also belong to the class of memory based algorithms.

The basic idea behind these methods is the use of the adaptive environment model in reinforcement learning. The efficiency of these approaches is much better than the one obtained for TD($\lambda > 0$), but it is followed by a considerable increase in the learning time. The increase in the learning time is caused by an additional update of the policy for states active in the past. Policy updates in the algorithms mentioned above are performed in the incremental mode, i.e., in each iteration of the algorithm. The advantage of such an approach is the immediate policy update. However, the application of methods which are related to the use of the information about states active in the past may lead to an increase in the algorithm's computational time.

The epoch policy update method is completely different. It is characterized by a suspension of the policy update up to the end of an episode. The experiences (series of states, actions and rewards) observed within the episode are stored directly or indirectly, i.e., in a processed form. The agent works on the basis of an unchanged policy in all steps of an episode. Once the last step of an episode is performed, the policy is modified by using stored experiences. A drawback of such a learning method is the lack of the policy update within the episode while the advantage is no computational burden. Epoch algorithms are not widely used due to a delay in the policy update (Reynolds, 2002; Rummery and Niranjan, 1994). The truncating temporal differences technique, in short TTD(λ) (Cichosz, 1995), is a example of the algorithm where a policy update delay takes place. In TTD(λ), the delay is set to some number of steps, and for the policy update a cumulative reward is used. The learning time of a single step is slightly longer than in the TD(0) method.

TTD(λ) is also applied by Reynolds (2002) in the algorithm that used the environment observation stored in a stack but the updates are performed in the epoch mode. For policy modification in the epoch mode one also uses simulated annealing (Atiya *et al.*, 2003), the tree based method (Ernst *et al.*, 2005), the temporal differences approach (Lagoudakis and Parr, 2003; Markowska-Kaczmar and Kwaśnicka, 2005), neural networks (Riedmiller, 2005) or genetic algorithms (Moriarty *et al.*, 1999) and evolutionary computation approaches (Whiteson, 2012). The suspension of the policy update, either for some number of iterations or until the end of the episode, causes the environment exploration to be performed on the basis of the policy which cannot follow the environment changes. This type of learning has one undisputed advantage: it offers a possibility of advanced processing of a series of interactions between the agent and the environment.

The epoch-incremental mode is a compromise between incremental and epoch learning modes. It allows modifying the policy in both the modes. In the literature, one can find two implementations of epoch-incremental algorithms. The first approach consists in the use of an agent experience in preliminary determination of the parameters of a learning system. For example, the solution proposed by Lin (1993), named ‘experience replay’, similarly to the temporal differences algorithm TD(λ) allows a single experience to update the state-action function of some states active in the past. For this purpose, supervised learning in the epoch mode is used for preliminary adjustment of the action-value function. Such a solution allows constraining the number of a real-object experiments. This idea is applied in the control of a mobile robot (Smart and Kaelbing, 2002; Ye *et al.*, 2003) or a car in a traffic (Forbes, 2002). However, this approach requires a training set. The second method of connecting the epoch mode algorithm and the incremental mode algorithm relies on the execution of the reinforcement learning algorithm in the incremental mode. In turn, in the epoch mode, the policy update is based on the stored experiences. Gelly and Silver (2007) proposed an epoch-incremental algorithm which, in the epoch mode, creates the optimal policy by means of the decision tree from the initial to the terminal state. An interesting idea can be found in the work of Vanhulsel *et al.* (2009), where the bucket brigade algorithm is applied. This approach is used to determine the order of state update on the basis of the temporal differences error TD. Moreover, there are also hybrid methods which combine coevolutionary algorithms and basic value-function approaches (Q-learning or TDL) (Krawiec *et al.*, 2011; Whiteson and Stone, 2006). These methods can be considered early examples of epoch-incremental learning algorithms.

In the environment where the rewards are located in the terminal states, it is possible to perform the policy update on the basis of a partial environment model after the terminal state is achieved. One can modify the policy elements which allow achieving the terminal state in the optimal way, i.e., according to the shortest path. In this paper, epoch-incremental reinforcement learning algorithms are proposed in order to obtain a highly efficient learning method with a small learning time. The main idea of these algorithms is to combine the Q-learning algorithm (either $Q(0)$-learning or $Q(\lambda)$-learning), which is performed in the incremental mode, with the acceleration method performed in the epoch mode. The proposed acceleration method is based, to a large extent, on the semi-optimal policy of states visited within a single episode. This semi-optimal strategy forms a basis for the modification of either the action-value function Q (in the algorithm that uses $Q(0)$-learning) or eligibility traces (in the algorithm that uses $Q(\lambda)$-learning). The proposed algorithms are compared with four well-known and often used reinforcement learning algorithms: $Q(0)$-learning, $Q(\lambda)$-learning, Dyna-Q and prioritized sweeping. All algorithms are used to solve the control problem of four grid worlds.

The article is organized as follows. Section 2 highlights reinforcement learning algorithms such as $Q(0)$-learning, $Q(\lambda)$-learning, Dyna-Q and prioritized sweeping. In Section 3, two epoch-incremental reinforcement learning algorithms are described. Section 4 compares $Q(0)$-learning, $Q(\lambda)$-learning, Dyna-Q, prioritized sweeping and two proposed epoch-incremental algorithms in the grid environment. In Section 5, results of experiments are shown. The work is concluded in Section 6.

2. Reinforcement learning

Reinforcement learning addresses the problem of an agent that must learn to perform a task through a trial-and-error interaction with an unknown environment. The agent and the environment interact until a terminal state is reached. The agent senses the environment through its sensors and, based on its current sensory inputs, selects an action to be performed in the environment. Depending on the effect of its action, the agent obtains a reward (Lanzi, 2000). It is necessary to note that an agent interacts in the environment which has the Markov property. An environment has the Markov property if the reward and the next state depend only on the current state and action. However, this dependency may be stochastic. Thus, $P^a_{ss'}$ defines the probability of the transition to the state $s' \in S$ after the execution of the action $a \in A$ in the state $s \in S$. A Markov Decision Process (MDP) in an environment is understood as the following quadruple: a set of states S, a set of actions A, a probability $P^a_{ss'}$ and a reward function r. The goal of the agent in the Markov environment is to maximize the discounted sum of future reinforcements r_t received in the long run, which is usually formalized as $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0,1]$ is the agent's discount rate.

2.1. *Q*-learning. There exist various types of reinforcement learning algorithms. Q-learning, proposed by Watkins (1989), is one of the most often used. This algorithm computes the table of all values $Q(s,a)$ (called the Q-table) by successive approximations. $Q(s,a)$ represents the expected payoff that the agent can obtain in state s after it performs action a. The Q-table is updated according to the following formula (Watkins, 1989):

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) \\ &\quad + \beta\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big), \end{aligned} \tag{1}$$

where the maximization operator refers to the action value a' which may be performed in the next state s' and $\beta \in (0,1]$ is the learning rate.

The basic Q-learning algorithm (1-step Q-learning) can be significantly improved considering the history of states' activation represented by eligibility traces. The eligibility trace is parametrized by the recency factor $\lambda \in [0,1]$, and therefore this enriched learning method is called $Q(\lambda)$-learning. Watkins (1989) proposed combining eligibility traces and Q-learning, which leads to the following update method:

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) \\ &\quad + \beta\Big(r + \gamma \max_{a'} Q(s',a') \\ &\quad - Q(s,a)\Big) e(s,a), \end{aligned} \tag{2}$$

where the eligibility trace for the state-action pair (s,a) is determined as

$$e(s,a) \leftarrow \gamma\lambda e(s,a) + \delta(s,a), \tag{3}$$

and

$$\delta(s,a) = \begin{cases} 1 & \text{for the active pair } (s,a), \\ 0 & \text{for all other pairs } (s,a). \end{cases} \tag{4}$$

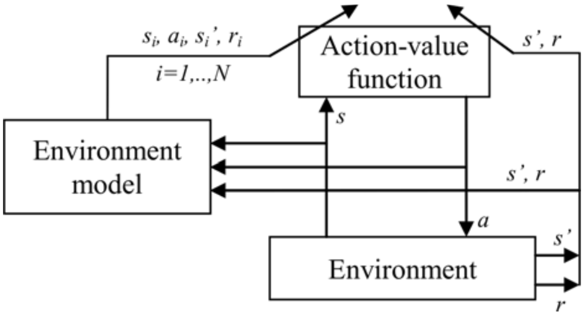


Fig. 1. Model based reinforcement learning.

Since $e(s,a)$ represents the eligibility trace of an action pair (s,a), its initial value is set to zero. This is because there is no action pair (s,a) active before the exploration of the environment. The eligibility trace of each pair (s,a) becomes large after state activation and then it decreases exponentially until the state is visited again.

2.2. Dyna-*Q* and prioritized sweeping. Dyna-Q and prioritized sweeping require a model of the environment in order to improve the policy represented by a Q-table. Given a state and an action, a model predicts the next state and the next reward. Dyna-Q is a $Q(0)$-learning based algorithm which simultaneously uses experience to build the model and to adjust the policy (Fig. 1). Additionally, it uses the model to adjust the policy. Dyna-Q operates in a loop of interactions with the environment as follows:

- Update the model after each transition $(s,a) \rightarrow (s',r)$. The model is stored in the table of state and action indices; the table predicts next state s' and reward r.
- Update the policy at the state s and the action a using the rule in (1).
- Perform N additional policy updates on the basis of the model. Each of these updates consists of a random choice of the state action pair (s_i, a_i) that has been experienced before and the query to the model with this pair. The model returns the next state s_i' and the reward r_i as its prediction. Finally, the update according to (1) is performed. A reasonable value of N can be determined on the basis of the relative speeds of computation and of taking action (Kaelbing *et al.*, 1996).

The prioritized sweeping algorithm is similar to the Dyna-Q, except that its updates are not chosen at random but depending on the priority which is the absolute value of the temporal difference error $|r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$. If this priority is greater than the arbitrary defined threshold, then the pair

(s, a) is stored into the queue with priority. Finally, only N pairs with the highest priority are updated according to (1) using the model, similarly to Dyna-Q (Sutton and Barto, 1998).

3. Epoch-incremental reinforcement learning algorithms

This section describes two proposed reinforcement learning algorithms executed in the epoch-incremental mode. The first algorithm, which uses $Q(0)$-learning, is called $EIQ(0)$-learning. The second one, which uses $Q(\lambda)$-learning, is called $EIQ(\lambda)$-learning. The $EIQ(0)$-learning algorithm with fuzzy approximation of the action-value function is described by Zajdel (2012). This section presents an implementation of this algorithm for a tabular representation of the action-value function. In both the algorithms, the agent–environment interactions take place in the incremental mode. These interactions are then used to modify the agent's policy represented by the action-value function Q (Fig. 2). The incremental mode is ended when the terminal state is reached and the reinforcement signal $r^{TERMINAL}$ is assigned. Then, the epoch mode begins where, on the basis of the environment model, the distances d from the terminal state are computed for all the states active in the past. Moreover, an action a, which allows reaching the terminal state in an optimal way, is assigned to each state. The shortest distance is meant to be the optimality criterion.

3.1. $EIQ(0)$-learning.

The epoch-incremental reinforcement learning algorithm which uses the $Q(0)$-learning method is executed in two stages. The first, incremental stage is performed until the terminal state is reached. In this mode, the agent policy is modified using the $Q(0)$-learning algorithm (1) and the environment model is created.

Algorithm 1 shows the instructions of the $EIQ(0)$-learning algorithm expressed in terms of the pseudocode. It starts with the initialization of the action-value function $Q(s, a)$ and the function $M(s, a, s')$ which is used to store the indicator of possible transitions from the state s to s' after the execution of the action a. Additionally, the queue $DQueue$ is set to be empty. This queue contains the following four elements: the state s, the optimal action a, the next state s' and the distance d from the terminal state (Step 1 of Algorithm 1).

In Step 2, a series of episodes begins. Each episode consists of the instructions performed in the incremental mode (Steps 3–9) and in the epoch mode (Steps 10–17) after the terminal state is reached. At the beginning of each episode, the state s is initialized. Then, the $Q(0)$-learning algorithm is executed (Steps 5–7). For the state s, where the execution of the action a results in the transition to the state s', the activity indicator $M(s, a, s')$ is set to 1. After the terminal state $s^{TERMINAL}$ is reached and the reinforcement signal $r^{TERMINAL}$ is assigned, the epoch mode begins. In the first step of the epoch mode, all four initial elements are inserted in $DQueue$. These four elements constitute a quadruple. This quadruple informs that the terminal state is reached and the distance between the current and the terminal state equals 0 (Step 10). $DQueue$ is realized as a priority queue where d acts as a priority parameter. The notation $DQueue_d$ is simply the set of quadruples for which the distance from the terminal states equals d.

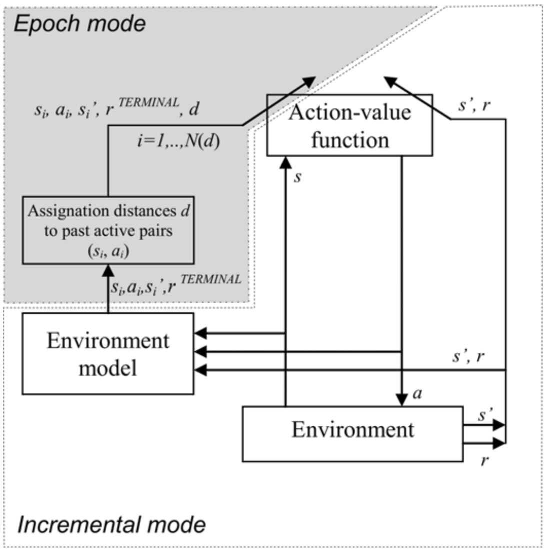

Epoch mode
s_i, a_i, s_i', $r^{TERMINAL}$, d
$i=1,..,N(d)$
Action-value function
s', r
s
Assignation distances d to past active pairs (s_i, a_i)
$s_i,a_i,s_i',r^{TERMINAL}$
Environment model
s', r
a
Environment
s'
r
Incremental mode

Fig. 2. Epoch-incremental reinforcement learning algorithm.

Since $DQueue$ is not empty (Step 11), the distance to the terminal state is incremented (Step 12). Then, the state-action pairs are searched to find the ones for which the next state is the terminal state (Steps 13–15). $DQueue_{d-1}$ denotes the set of 4-tuples $(s, a, s', d-1)$ in which the distance from the terminal states equals $d-1$. $s' \in DQueue_{d-1}$ denotes the next state s' (i.e., the third element of $DQueue_{d-1}$) for which the distance is $d-1$. Thereafter, the action-value function is updated as follows:

$$Q(s, a) \leftarrow Q(s, a) + \beta(r^{TERMINAL}(\gamma\lambda)^d + \gamma \max_{a'} Q(s', a') - Q(s, a)), \tag{5}$$

where d is the shortest distance between the actual and the terminal state, and $\lambda \in (0, 1]$ is the recency factor. The multiplication of the reinforcement signal $r^{TERMINAL}$ by the $(\gamma\lambda)^d$ coefficient realizes exponentially decreasing dependence between states active in the past and the terminal state.

The $\gamma\lambda$ product was first introduced in the TD(λ) algorithm in which the eligibility trace of state-action pairs is decreased by this factor in each iteration. In

Algorithm 1. $EIQ(0)$-learning.

```
 1. Initialize $Q(s,a)$ and $M(s,a,s')=0$ for all $s,s' \in S$, $a \in A$ and $DQueue$ to empty
 2. foreach episode:
 3.    Initialize $s$
 4.    repeat (for each step of episode):
 5.       Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-$greedy(s,Q)$
 6.       Execute action $a$, observe resultant state $s'$ and reward $r$
 7.       $Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
 8.       $M(s,a,s') = 1$
 9.    until $s = s^{TERMINAL}$
10.    $DQueue_0 \leftarrow (s,a,s^{TERMINAL}, d=0)$
11.    while $DQueue_d$ is not empty
12.       $d \leftarrow d+1$
13.       foreach $s' \in DQueue_{d-1}$
14.          foreach $s \in preds(s')$ and $a \in A$:
15.             if $M(s,a,s')=1$
16.                $DQueue_d \leftarrow (s,a,s',d)$
17.                $Q(s,a) \leftarrow Q(s,a) + \beta(r^{TERMINAL}(\gamma\lambda)^d + \gamma \max_{a'} Q(s',a') - Q(s,a))$
18.       remove $DQueue_{d-1}$
```

the TD(λ) method, all the elements of the action-value function are updated depending on the distance from the actual state. In the case of the proposed epoch-incremental algorithm, the updates are performed only for these elements of the action-value function Q which are responsible for the suboptimal strategy determined on the basis of the shortest distance to the absorbing state. After the update of the action-value function for the pairs (s,a) for which the next state is the terminal state, the initializing quadruple (Step 10) is removed from $DQueue$. At this stage of the algorithm, $DQueue$ contains only quadruples with $d = 1$. The algorithm iterates from Step 12 until (i) the shortest distance to the terminal state is assigned to all active states within a current episode and (ii) the action-value function with reinforcement signal $r^{TERMINAL}$ is updated according to the formula (5). It is worth noting that Steps 10–16 of Algorithm 1 can be actually considered a BFS (Breadth-First Search) through the graph of state transitions, starting from the terminal state.

3.2. $EIQ(\lambda)$-learning.

$EIQ(\lambda)$-learning is an epoch-incremental reinforcement learning algorithm which uses the $Q(\lambda)$-learning method. In order to improve the agent policy, this algorithm uses the distance d, in much the same way as $EIQ(0)$-learning. In the incremental mode, the $Q(\lambda)$-learning algorithm is performed and, therefore, the eligibility traces are applied (3) (Steps 7–12 in Algorithm 2). In the last step of the incremental mode, the activity indicator $M(s,a,s')$ of a 3-tuple (s,a,s') is set to 1. As in the case of $EIQ(0)$-learning (Algorithm 1), the epoch part starts when the terminal state is reached. $DQueue$ is initialized (Step 15) and the eligibility traces are set to zero (Step 16). In Watkins' (1989) $Q(\lambda)$-learning algorithm, the eligibility traces are set to zero at the beginning of each episode.

The eligibility traces are some kind of memory of state–action pairs active in the past. In the proposed algorithm, the eligibility traces, at the beginning of each episode, represent the suboptimal policy obtained from the previous episode. Similarly to $EIQ(0)$-learning, one computes the shortest distance d to the terminal state of the states active in the past (starting from step 17). The important difference between $EIQ(\lambda)$-learning and $EIQ(0)$-learning is the way of utilizing d. In the $EIQ(0)$-learning algorithm, the distance d to the terminal state is used in the backpropagation of the reinforcement signal $r^{TERMINAL}$ (5). In the algorithm described in this section, the update of the action-value function is performed throughout the initialization of the eligibility traces before each episode in the following way:

$$e(s,a) = \begin{cases} (\gamma\lambda)^d & \text{if } M(s,a,s')=1, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

In this way, the eligibility traces which are nearest to the terminal state ($d = 1$) are set to the highest value equal to $\gamma\lambda$. Along with the increase in the distance d, the initial values of $e\,(s,a)$ are the numbers that decrease exponentially.

Algorithm 2. $EIQ(\lambda)$-learning.

```
Initialize $Q(s,a)$, $e(s,a)=0$ and $M(s,a,s')=0$ for all $s,s' \in S$, $a \in A$ and $DQueue$ to empty
foreach episode:
  Initialize $s$
  repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon\text{-}greedy(s,Q)$
    Execute action $a$, observe resultant state $s'$ and reward $r$
    foreach $(s,a)$
      $e(s,a)=\gamma\lambda e(s,a)$
    for actual $(s,a)$
      $e(s,a) \leftarrow e(s,a)+1$
    foreach $(s,a)$
      $Q(s,a) \leftarrow Q(s,a)+\beta(r+\gamma \max_{a'} Q(s',a')-Q(s,a))e(s,a)$
    $M(s,a,s')=1$
  until $s=s^{TERMINAL}$
  $DQueue_0 \leftarrow (s,a,s^{TERMINAL},d=0)$
  Initialize $e(s,a)=0$ for all $s \in S$ and $a \in A$
  while $DQueue_d$ is not empty
    $d \leftarrow d+1$
    foreach $s' \in DQueue_{d-1}$:
      foreach $s \in preds(s')$ and $a \in A$:
        if $M(s,a,s')=1$
          $DQueue_d \leftarrow (s,a,s',d)$
          $e(s,a)=(\gamma\lambda)^d$
    remove $DQueue_{d-1}$
```

4. Case study

The idea of the epoch-incremental reinforcement learning algorithms is the result of the observation of the performance of $Q(0)$-learning, $Q(\lambda)$-learning, Dyna-Q and prioritized sweeping algorithms in grid worlds. Let us consider the environment GRID69 shown in Fig. 3(a) (Sutton, 1990). This is a stationary environment with the initial state 'Start' (State 19) and the single absorbing state 'Stop' (State 9). The states marked as dark gray squares represent obstacles and are unreachable for an agent. In this environment, in each of reachable states, the agent can perform one of four actions: ←, →, ↑ and ↓. When reaching the absorbing state (the execution of action ↑ in State 18 and the transition to State 9), the agent is assigned the reinforcement signal equal to 1 and the episode is finished. The reinforcement signal in the remaining states is 0 (Fig. 3(a)).

Let us consider the performance of the agent in the first episode. The agent policy is represented by the action-value function Q. Let us also assume that this function is initialized with 0 values for all state–action pairs. Therefore, the agent does not possess any information related to the way of how to reach the goal. Moreover, since the reinforcement signal is zero, the action-value function is not updated, either (see (1) and (2)). The agent only performs the exploration of the environment. Figure 3(b) presents the actions which might be performed in the first episode before the absorbing state is reached. The real experiments show that such a situation is unlikely to occur. Within the first episode, the agent usually performs 95%–100% of actions in all grid states. A hypothetical situation presented in Fig. 3(b) will highly facilitate further analysis. Let us also assume that the agent reaches the absorbing state by moving along the path illustrated in Fig. 3(c). Taking into account the actions shown in Fig. 3(b), the agent moves along this path in both directions (either to or from the state 'Stop'). The transition to the absorbing State 9 results in receiving a reward equal to 1. The type of learning algorithm determines the way of policy update.

In the $Q(0)$-learning algorithm, only $Q(18,\uparrow)$ is updated (Fig. 3(d)). In the $Q(\lambda)$-learning algorithm, only these elements of Q-table are modified for which the appropriate state–action pairs were active in the past ($e(s,a) > 0$). It is worth noting that not only $Q(s,a)$ elements responsible for the transition to the absorbing state (e.g., $Q(18,\uparrow)$ are modified, but also the

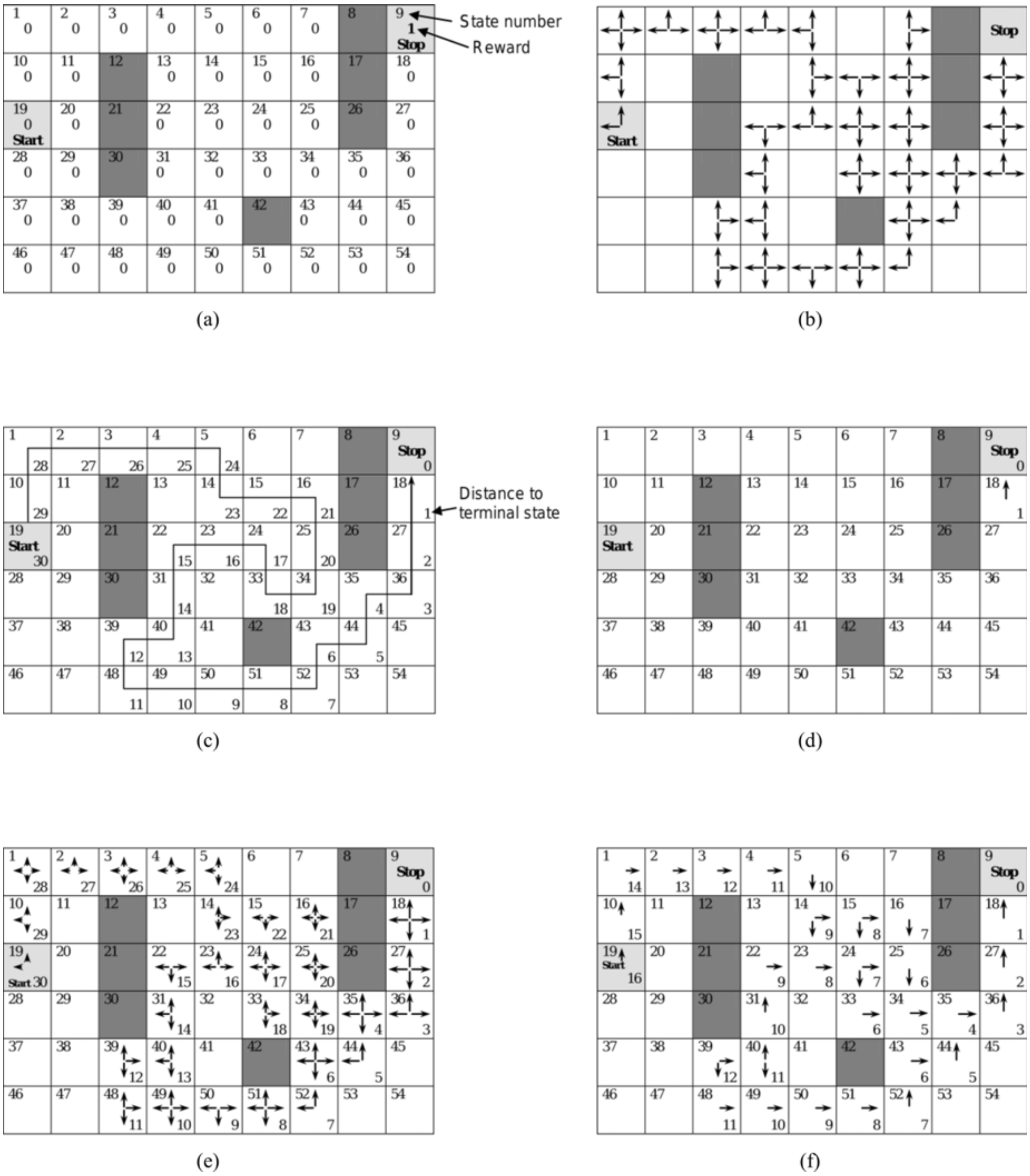


Fig. 3. GRID69 environment: numbers of states and rewards (a), actions performed within the episode considered (b), sample path of transitions from the 'Start' state to the absorbing 'Stop' state (c), presentation of the modified elements of the action-value function for the algorithms: $Q(0)$-learning (d), $Q(\lambda)$-learning (e), $EIQ(0)$-learning and $EIQ(\lambda)$-learning (f).

elements which correspond to non-optimal actions (e.g., $Q(18,\leftarrow)$, $Q(18,\rightarrow)$, $Q(18,\downarrow)$). Let us also pay attention to the fact that in State 34 all four actions are performed. The action ← is executed as the last one. It causes transition to State 33 and increases the distance from the goal (Figs. 3(b) and (c)). The $Q(\lambda)$-learning algorithm does not use the information whether the agent has visited State 35, which is placed much closer to the terminal state. This algorithm enhances "the path of last executed or most active" state–action pairs. The distance from the absorbing state 'Stop' (Fig. 3(c)) is marked in the bottom-right corner of the squares which correspond to the states active in the past. In the first episode, this distance results from the transition path determined only throughout the environment exploration.

Exponentially decreasing values of the eligibility traces are functions of these distances (Fig. 3(e)). One can notice that State 35 is located four transitions away from the terminal state (Fig. 3(c)). The neighbouring State 34 lies nineteen transitions away from the terminal state. The use of information that the transition from State 34 to 35 after the execution action → ($M(34,\rightarrow,35) =$

1) would improve the agent policy. Epoch-incremental reinforcement learning algorithms use such information. In the epoch mode of these algorithms, one determines the pair $(34, \rightarrow)$ as the candidate for learning. Moreover, the shorter distance equal to 5 is assigned to State 34. Such a distance results from the states' neighbourhood represented by the function $M(s, a, s')$. On the basis of this information, the element of the action-value function $Q(34, \rightarrow)$ is updated either in $EIQ(0)$-learning or throughout $e(34, \rightarrow)$ in $EIQ(\lambda)$-learning. As a result of the aforementioned procedure, out of all actions executed in states active in the past only these actions are selected which realize the suboptimal policy in the sense of the shortest path, as shown in Fig. 3(f). Thus, after the first episode, the agent policy is updated with the use of a suboptimal action.

5. Experiments

Grid worlds are often used for comparing how quickly different reinforcement learning methods converge to a stable solution (Cichosz, 1995; Crook and Hayes, 2003; Forbes, 2002; Gelly and Silver, 2007; Moore and Atkeson, 1993; Peng and Williams, 1993; Reynolds, 2002; Sutton, 1990; Sutton and Barto, 1998). This chapter presents the results of experiments conducted in two types of grid environments which model *task to success* and *task to failure*. In *task to success*, the purpose of the update of the agent policy is to reach the terminal state as quickly as possible. When the agent moves to the terminal state, it receives the reinforcement signal $r^{TERMINAL}$ greater than in previous states, and hence $r^{TERMINAL} > r$. *Task to failure*, in turn, is a type of episodic task where the goal for an agent is to avoid certain undesirable states. The achievement of such a state denotes a failure and ends the episode. In such a type of task, the achievement of absorbing states is rewarded with the reinforcement signal $r^{TERMINAL} < r$.

5.1. Task to success. The experiments are performed in GRID69 and GRID2436 grid worlds presented by Sutton (1990) as well as Peng and Williams (1993), respectively. GRID2436 contains 24 rows and 36 columns (Fig. 4) extended form of the GRID69 environment shown in Fig. 3. The minimal number of transitions (states' changes) from the starting state ('Start') to the terminal state ('Stop') equals 16 in GRID69 and 58 in GRID2436. The number of reachable states in GRID69 and GRID2436 is 47 and 752, respectively.

$EIQ(0)$-learning and $EIQ(\lambda)$-learning algorithms are compared with $Q(0)$-learning, $Q(\lambda)$-learning, Dyna-Q and prioritized sweeping. The initial values of $Q(s, a)$ and $M(s, a, s')$ are zero. For all the algorithms one assumes the learning rate $\alpha = 0.1$, the discount rate $\gamma = 0.95$ and the recency factor $\lambda = 0.2$. The values of these coefficients are chosen by a trial-and-error method, which provided the best performance among all tested algorithms. The exploration policy parameter ϵ is set to 0.1. Moreover, in Dyna-Q and prioritized sweeping algorithms, the number of the updates based on the environment model is assumed to be 50.

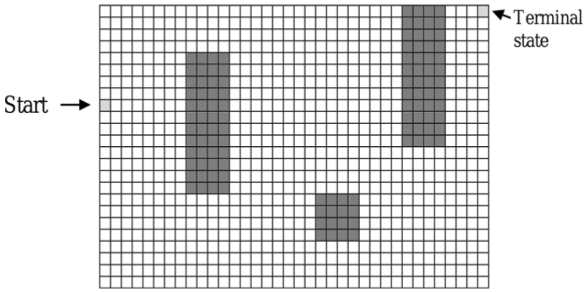


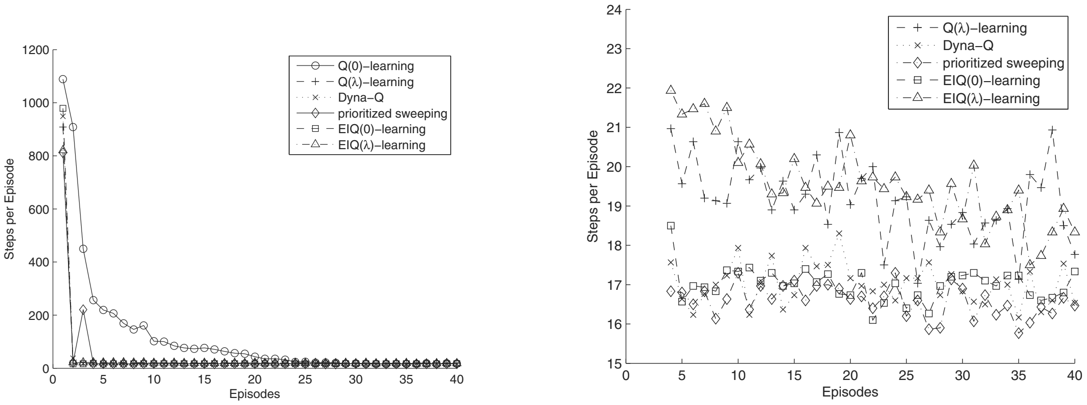
Fig. 4. Grid world GRID2436.

Figures 5(a) and 6(a) show the averaged (over 30 repetitions) number of the transitions required to reach the terminal state 'Stop'. One can notice that the convergence of the $Q(0)$-learning algorithm is slightly worse in the case of GRID69 and much worse for GRID2436 in comparison with other algorithms. It is worth keeping in mind that the minimal number of transitions for this environment is 58.

In Figs. 5(b) and 6(b), the results for five most efficient algorithms are shown. In order to illustrate the difference between the efficiencies of the algorithms, the results are shown starting from the fourth episode. In the case of the GRID69 environment one can distinguish two groups of plots. $Q(\lambda)$-learning and $EIQ(\lambda)$-learning are visibly worse than the remaining ones, i.e., Dyna-Q, prioritized sweeping and $EIQ(0)$-learning. In the case of GRID2436, the use of the information about the distance between all the states and the absorbing state as well as the initialization of the eligibility traces in the epoch mode (realized in $EIQ(\lambda)$-learning algorithm) increase the efficiency of the $Q(\lambda)$-learning algorithm. In both the environments, the $EIQ(0)$-learning algorithm performs much better than the $EIQ(\lambda)$-learning method.

In GRID2436 the number of steps per episode is smaller for $EIQ(0)$-learning in comparison with Dyna-Q and prioritized sweeping algorithms only for the fourth and the fifth episode. Along with the increase in the number of the episodes, the $EIQ(0)$-learning algorithm requires a greater (compared with Dyna-Q and prioritized sweeping) number of steps to reach the terminal state. The average number of steps (μ) and its standard deviation (σ) for the first 3 episodes, the average for all 40 episodes and the average for the last 30 episodes are presented in Tables 1 and 2.

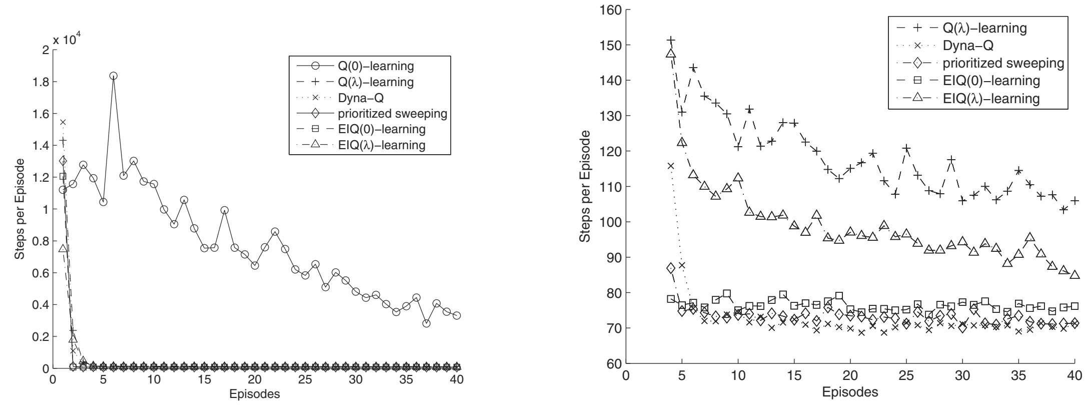
Task to success modelled by the GRID69 environment turns out to be relatively undemanding.



(a)

(b)

Fig. 5. GRID69 environment: steps from the first (a) and the fourth (b) episode.



(a)

(b)

Fig. 6. GRID2436 environment: steps from the first (a) and the fourth (b) episode.

Table 1. GRID69 environment: number of steps necessary to reach the terminal ‘Stop’ state starting from the ‘Start’ state.

Episodes	Number of steps											
	$Q(0)$-learning		$Q(\lambda)$-learning		Dyna-Q		prioritized sweeping		$EIQ(0)$-learning		$EIQ(\lambda)$-learning	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
1	1089	942	909	637	949	864	811	681	979	439	821	415
2	908	842	20	4	38	9	22	4	17	2	22	5
3	450	342	21	5	18	4	221	455	17	3	22	5
Average for all episodes (1–40)	121	94	41	20	41	25	42	31	34	14	32	14
Average for last 30 episodes (11–40)	32	24	19	5	17	3	17	2	17	2	19	4

Except for $Q(0)$-learning, all the remaining algorithms, already in the second episode, achieve the number of steps one order smaller than in the first episode. For GRID2436, the number of steps for first three episodes is almost constant for $Q(0)$-learning. For the remaining algorithms, already in the third episode, the number of

Table 2. GRID2436 environment: number of steps necessary to reach the terminal ‘Stop’ state starting from the ‘Start’ state.

Episodes	Number of steps											
	$Q(0)$-learning		$Q(\lambda)$-learning		Dyna-Q		prioritized sweeping		$EIQ(0)$-learning		$EIQ(\lambda)$-learning	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
1	11206	11773	14311	11044	15452	16808	13022	10323	12051	9862	7474	6302
2	11572	13133	2380	2380	1098	230	116	15	78	15	1794	1269
3	12773	10533	402	452	283	155	73	10	80	22	458	407
Average for all episodes (1–40)	7793	6650	537	373	488	438	398	268	376	261	335	221
Average for last 30 episodes (11–40)	6406	5322	114	25	70	7	73	9	76	18	95	18

Table 3. GRID69 and GRID2436 environments: average times of a single iteration in the incremental and the epoch mode. The simulations were performed on a computer with an Intel Pentium M 2.0 GHz processor and 1 GB RAM, the algorithms were implemented in Matlab.

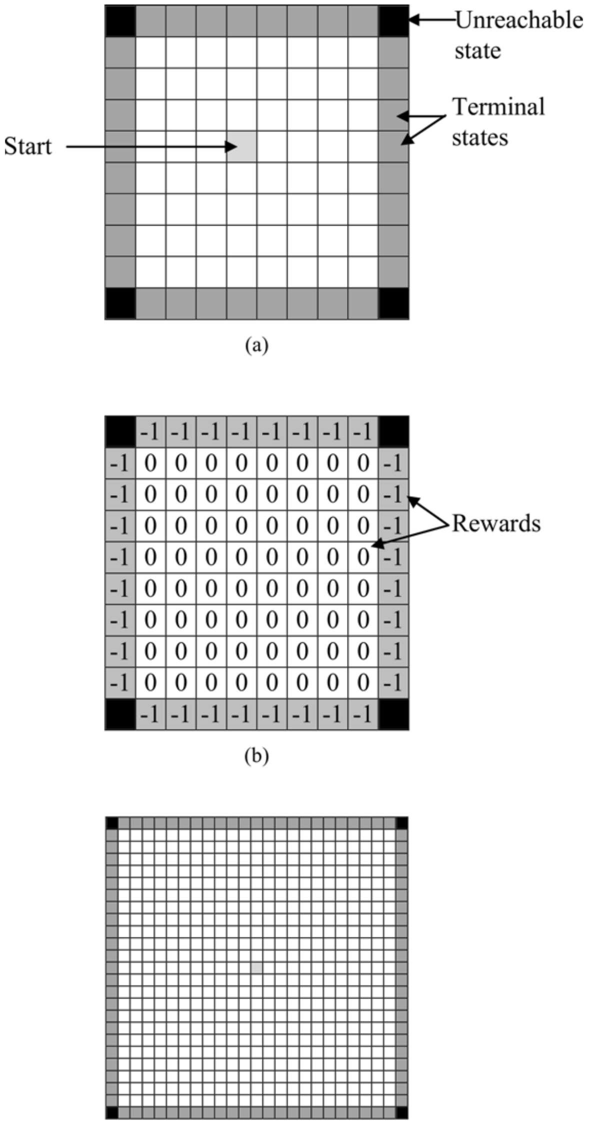
Environment	Mode	Time [ms]					
		$Q(0)$-learning	$Q(\lambda)$-learning	Dyna-Q	Prioritized sweeping	$EIQ(0)$-learning	$EIQ(\lambda)$-learning
GRID69	incremental	0.07	0.07	0.86	6.07	0.08	0.10
	epoch	–	–	–	–	3.56	2.55
GRID2436	incremental	0.08	0.11	0.91	119.88	0.10	0.12
	epoch	–	–	–	–	652.61	603.42

steps is of at least one order smaller than in the first episode. In the proposed $EIQ(0)$-learning algorithm, the number of steps decreases in the fastest way. For the prioritized sweeping algorithm, a decrease in the number of steps to 73 occurs in the third episode. The standard deviation in the first episode computed for all algorithms is large and of the order of μ. This is because the environment exploration is performed on the basis of a pure stochastic policy. In the next stages, a decrease of σ is an indicator of a semioptimal policy creation degree. Analysing the average number of steps for all 40 episodes, one can notice that the $EIQ(\lambda)$-learning algorithm seems to be the best method. It is influenced by a relatively small number of transitions in the first episode. Hence, discarding the initial episodes makes the average number of transitions for 30 last episodes (bottom row in Tables 2 and 3) the most reliable measure of efficiency. As can be observed, for GRID2436, only Dyna-Q, prioritized sweeping and $EIQ(0)$-learning algorithms have the number of transitions smaller than 80, 70, 73 and 76, respectively. In the case of GRID69, only for $Q(0)$-learning, the average number of episodes much higher than for the remaining methods.

It is necessary to pay attention to the computational cost of such a result. In Table 3, the execution times of each compared algorithm are presented. For epoch-incremental algorithms, the times of the incremental and epoch mode are shown. The times of the incremental modes for $EIQ(0)$-learning and $EIQ(\lambda)$-learning algorithms are slightly longer than the corresponding results of $Q(0)$-learning and $Q(\lambda)$-learning. This results from the necessity of building the environment model represented by $M(s, a, s')$. The execution times of Dyna-Q and prioritized sweeping algorithms are longer than the times of the algorithms mentioned above. This is a result of the action-value function update which is based on the environment model. The execution time of the prioritized sweeping algorithm is particularly long since in this method the states’ sorting is applied according to the absolute value of the temporal differences error $|TD|$. Moreover, for these sorted states, one determines related state–action pairs (Moore and Atkeson, 1993; Sutton and Barto, 1998). The computational time of the epoch mode of $EIQ(0)$-learning and $EIQ(\lambda)$-learning is long but this mode is performed after the episode ends. Therefore it is executed rarely in comparison with the incremental mode.

As one can observe, the use of the environment model in the epoch mode leads to improvements in both the $Q(0)$-learning and the $Q(\lambda)$-learning algorithm. However, much better results are achieved in the case of the $EIQ(0)$-learning algorithm by the direct interference in the action-value function. The modification of the eligibility traces in the $EIQ(\lambda)$-learning algorithm improves the efficiency of the $Q(\lambda)$-learning method, but this improvement is not so spectacular, in contrast to the $EIQ(0)$-learning algorithm.

5.2. Task to failure. A sample 10×10 grid environment (denoted henceforth as GRID1010) modelling *task to failure* is presented in Fig. 7(a). The environment has the initial 'Start' state, 32 terminal states located at the edge of the environment and 4 unreachable states (in the corners of the grid).



	-1	-1	-1	-1	-1	-1	-1	-1	
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	-1
	-1	-1	-1	-1	-1	-1	-1	-1	

Fig. 7. GRID1010 environment: starting state, unreachable and terminal states (a), reinforcement signals for the particular states (b), GRID2525 environment (c).

Figure 7(c) shows GRID2525, which is an extension of the GRID1010 environment. The achievement of the terminal states is related to the assignment of the reinforcement signal $r^{TERMINAL} = -1$ and ends the episode. The purpose of the learning algorithm is to keep the agent within the white-field square as long as possible. Then, the agent is assigned the reinforcement signal $r = 0$ (Fig. 7(b)). It is assumed that the system learns to avoid the terminal states if it performs 1000 steps inside the area of white fields. For each of the compared algorithms, the learning process is repeated 10 times.

The averaged results are presented in Table 4. The number of episodes before the agent learned to avoid terminal states is a measure of the efficiency of the algorithms. The smaller the number in Table 4, the more efficient the learning algorithm. In both the environments, the application of epoch-incremental algorithms decreases the number of episodes necessary to achieve a stable policy. In the GRID2525 environment, the number of episodes for all algorithms is similar. However, the GRID1010 environment is more demanding. The advantages of using epoch-incremental algorithms are more noticeable. The $EIQ(0)$-learning algorithm provides the best results since it only requires 3.3 episodes to learn how to avoid the terminal states. For $EIQ(\lambda)$-learning and $Q(\lambda)$-learning algorithms 4.0 and 7.6 episodes, respectively are merely needed. For $Q(0)$-learning, Dyna-Q and prioritized sweeping methods, the number of episodes exceeds 26.

6. Conclusions

This article proposed a new class o reinforcement learning algorithms: $EIQ(0)$-learning and $EIQ(\lambda)$-learning. These algorithms are based on the environment model and the distance from the absorbing state.

The novelty introduced to the proposed methods consisted in the improvement of the agent policy on the basis of the model in the epoch mode. Such a solution shortened the execution time of the instructions in the incremental mode in comparison to Dyna-Q or prioritized sweeping algorithms. Nonetheless, the efficiencies of $EIQ(0)$-learning, $EIQ(\lambda)$-learning, Dyna-Q and prioritized sweeping algorithms were comparable in the discussed *task to success*. In the examined *task to failure*, the efficiency of the proposed algorithms was the highest. The results of the experiments confirmed the validity of proposed modifications.

Another novelty presented in this contribution is the use of the environment model in the creation of a suboptimal policy. Furthermore, the environment model was utilized in the determination of the shortest distances between past-active states and the terminal state.

The author will adapt the proposed algorithms in a problem of mobile robot control, a cart pole system and a ball-beam system. The use of these algorithms in a

Table 4. GRID1010 and GRID2525 environments: average number of episodes required to learn how to avoid the terminal states.

Environment	Average number of episodes											
	$Q(0)$-learning		$Q(\lambda)$-learning		Dyna-Q		Prioritized sweeping		$EIQ(0)$-learning		$EIQ(\lambda)$-learning	
	μ	σ	μ	σ	μ	σ w	μ	σ	μ	σ	μ	σ
GRID1010	26.4	1.8	7.6	2.5	26.5	2.5	26.3	2.0	3.3	1.6	4.0	1.0
GRID2525	7.5	1.4	5.1	1.3	7.7	1.3	7.1	1.4	4.9	1.6	3.2	0.9

continuous-state environment will require the application of function approximators, e.g., a Takagi–Sugeno system, a fuzzy CMAC or a radial basis function neural network.

Acknowledgment

This work was supported by the Polish Ministry of Science and Higher Education under the grant N N516 374536.

References

Atiya, A.F., Parlos, A.G. and Ingber, L. (2003). A reinforcement learning method based on adaptive simulated annealing, *Proceedings of the 46th International Midwest Symposium on Circuits and Systems, Cairo, Egypt*, pp. 121–124.

Barto, A., Sutton, R. and Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning problem, *IEEE Transactions on Systems, Man, and Cybernetics* **13**(5): 834–847.

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of $TD(\lambda)$ for reinforcement learning, *Journal of Artificial Intelligence Research* **2**: 287–318.

Crook, P. and Hayes, G. (2003). Learning in a state of confusion: Perceptual aliasing in grid world navigation, *Technical Report EDI-INF-RR-0176*, University of Edinburgh, Edinburgh.

Ernst, D., Geurts, P. and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning, *Journal of Machine Learning Research* **6**: 503–556.

Forbes, J. R. N. (2002). *Reinforcement Learning for Autonomous Vehicles*, Ph.D. thesis, University of California, Berkeley, CA.

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in UCT, *Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, USA*, pp. 273–280.

Kaelbing, L.P., Litman, M.L. and Moore, A.W. (1996). Reinforcement learning: A survey, *Journal of Artificial Intelligence* **4**(1): 237–285.

Krawiec, K., Jaśkowski, W.G. and Szubert, M.G. (2011). Evolving small-board Go players using coevolutionary temporal difference learning with archives, *International Journal of Applied Mathematics and Computer Science* **21**(4): 717–731, DOI: 10.2478/v10006-011-0057-3.

Lagoudakis, M. and Parr, R. (2003). Least-squares policy iteration, *Journal of Machine Learning Research* **4**: 1107–1149.

Lanzi, P. (2000). Adaptive agents with reinforcement learning and internal memory, *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior, Cambridge, MA, USA*, pp. 333–342.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Markowska-Kaczmar, U. and Kwaśnicka, H. (2005). *Neural Networks Applications*, Wrocław University of Technology Press, Wrocław, (in Polish).

Moore, A. and Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less time, *Machine Learning* **13**(1): 103–130, DOI: 10.1007/BF00993104.

Moriarty, D., Schultz, A. and Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning, *Journal of Artificial Intelligence Research* **11**: 241–276.

Peng, J. and Williams, R. (1993). Efficient learning and planning within the Dyna framework, *Adaptive Behavior* **1**(4): 437–454.

Reynolds, S. (2002). Experience stack reinforcement learning for off-policy control, *Technical Report CSRP-02-1*, University of Birmingham, Birmingham, `ftp://ftp.cs.bham.ac.uk/pub/tech-reports/2002/CSRP-02-01.ps.gz`.

Riedmiller, M. (2005). Neural reinforcement learning to swing-up and balance a real pole, *Proceedings of the IEEE 2005 International Conference on Systems, Man and Cybernetics, Big Island, HI, USA*, pp. 3191–3196.

Rummery, G. and Niranjan, M. (1994). On-line q-learning using connectionist systems, *Technical Report CUED/F-INFENG/TR 166*, Cambridge University, Cambridge.

Smart, W. and Kaelbing, L. (2002). Effective reinforcement learning for mobile robots, *Proceedings of the International Conference on Robotics and Automation, Washington, DC, USA*, pp. 3404–3410.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, *Proceedings of the Seventh International Conference on Machine Learning, Austin, TX, USA*, pp. 216–224.

Sutton, R. (1991). Planning by incremental dynamic programming, *Proceedings of the 8th International Workshop on Machine Learning, Evanston, IL, USA*, pp. 353–357.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Vanhulsel, M., Janssens, D. and Vanhoof, K. (2009). Simulation of sequential data: An enhanced reinforcement learning approach, *Expert Systems with Applications* **36**(4): 8032–8039.

Watkins, C. (1989). *Learning from Delayed Rewards*, Ph.D. thesis, Cambridge University, Cambridge.

Whiteson, S. (2012). Evolutionary computation for reinforcement learning, *in* M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State of the Art*, Springer, Berlin, pp. 325–358.

Whiteson, S. and Stone, P. (2006). Evolutionary function approximation for reinforcement learning, *Journal of Machine Learning Research* **7**: 877–917.

Ye, C., Young, N.H.C. and Wang, D. (2003). A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **33**(1): 17–27.

Zajdel, R. (2012). Fuzzy epoch-incremental reinforcement learning algorithm, *in* L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L.A. Zadeh and J.M. Zurada (Eds.), *Artificial Intelligence and Soft Computing*, Lecture Notes in Computer Science, Vol. 7267, Springer-Verlag, Berlin/Heidelberg, pp. 359–366.

Roman Zajdel received the M.Sc. degree in electrical engineering from the Rzeszów University of Technology in 1990 and the Ph.D in computer science from the Wrocław University of Technology in 1999. He is an assistant professor at the Institute of Control and Computer Engineering, Rzeszów University of Technology. His research interests concentrate on reinforcement learning, fuzzy logic and neural networks.

Received: 10 July 2012
Revised: 13 March 2013
Re-revised: 20 June 2013