An Experience Replay Method based on Tree Structure for Reinforcement Learning

Wei-Cheng Jiang, Kao-Shing Hwang, Senior Member, IEEE, Jin-Ling Lin

Abstract—Q-Learning, which is a well-known model-free reinforcement learning algorithm, a learning agent explores an environment to update a state-action function. In reinforcement learning, the agent does not require information about the environment in advance, so an interaction between the agent and the environment is for collecting the real experiences that is also an expensive and time-consuming process. Therefore, to reduce the burden of the interaction, sample efficiency becomes an important role in reinforcement learning. This study proposes an adaptive tree structure integrating with experience replay for Q-Learning, called ERTS-Q. In ERTS-Q method, Q-Learning is used for policy learning, a tree structure establishes a virtual model which perceives two different continuous states after each state transaction, and then the variations of the continuous state are calculated. After each state transition, all states with highly similar variation are aggregated into the same leaf nodes. Otherwise, new leaf nodes will be produced. For experience replay, the tree structure predicts the next state and reward based on the statistical information that is stored in tree nodes. The virtual experiences produced by the tree structure are used for achieving extra learning. Simulations of the mountain car and a maze environment are performed to verify the validity of the proposed modeling learning approach.

Index Terms—Reinforcement learning, Dyna-Q architecture, Tree structure, Experience replay.

1 INTRODUCTION

Recently, autonomous robots have increasingly attracted and developed for intelligent robots. The learning methods for intelligent robots can be divided into three categories—supervised learning [1], unsupervised learning [2], and reinforcement learning (RL) [3]. Meanwhile, the RL can deal with many control problems without any knowledge about the dynamics [3]-[5]. The RL agent needs to interact with the environment with a sequence of the trial-and-error process. They collect interactive experiences to update the action-value function during the period of learning. In the process of interacting, reinforcement signals returned from the environment are used to strengthen or weaken the policy. In RL, the learning methods are roughly divided into two types, model-free RL and model-based RL. In model-free RL, the sensory input information perceived from the environment is generalized for updating the action-value function [6][7]. However, how to decide the resolution of the state space, which will influence the accuracy for the controller, is still an imperative issue. Each collecting experiences will be discarded after updating action-value function. However, collecting experiences is a very expensive and time-consuming process, especially when the state space is very large [8][9]. To use these collecting experiences efficiently, the concept of sample efficiency uses few actions to learn a good policy [10][11] in many RL applications.

Model-based RL is used to achieve sample efficiency, so it obtains high performance for policy learning [12]. Dyna-Q architecture is a well-known model-based RL that handles a virtual model which is similar to the environment. The model includes two functions, one is a transition function, and another is a reward function. In model-based RL, an agent collects experiences to update the two functions that are called model learning. The agent takes these two functions to produce extra simulated experiences for updating the action-value function, called planning [8]. However, Dyna architecture does not point out the structure of the model, so table structures widely take into consideration [13]-[16]. The table structure resolves the problem of the resolution of each cell for the continuous domain. Besides, the continuous state space aggregated into several cells with the same size is a straightforward method for modeling the environment. However, finding a way to set up the resolution for the table structure in a continuous space is another important issue [17]. A higher resolution structure is more accurate to approximate the environment, but it requires more learning time to explore the environment and collect experiences. However, the structure with lower resolution costs less time for learning, but it is not accurate for modeling the environment.

Other methods can be used to achieve the sampling efficiency, such as Experience Replay [18], Batch Reinforcement Learning [19] and Backward Q-Learning [20]. In the process of learning, a table must record all collecting experiences. However, there are infinite experiences in the continuous state space, but the size of the table structure is limited. So, some experiences must be omitted, when the number of collecting experiences is over the

- *Wei-Cheng Jiang is with the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. E-mail: enjoysea0605@gmail.com.*
- *Kao-Shing Hwang is with the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. He is also an adjunct professor of the Department of Healthcare Administration and Medical Informatic, Kaohsiung Medical University, Taiwan. E-mail: hwang@ccu.edu.tw (Corresponding author: Kao-Shing Hwang).*
- *Jin-Ling Lin is with the Department of Information Management, Shih Hsin University, Taipei 116, Taiwan. E-mail: jllin@mail.shu.edu.tw.*

size of the table. In [18], two omitted mechanisms, an updating based on time and an updating based on reward, are introduced to solve this problem. In the updating based on time, the mechanism omits the oldest collecting experience at first. In the updating based on reward, the mechanism searches for an experience with the lowest reward in the table and then compares it with the new collecting experience. The experience with a larger reward will be stored in the table. In [20], an agent uses SARSA-learning to learn the policy and records experience with the collected order in the current episode. When the agent meets a terminal state, the experiences are retrieved following the backward order and are used for *Q*-function to update the policy. In experience replay methods, the size of the table should be determined in advance. If the size is too large, many experiences can be recorded and reused. But it needs to cost much memory space. If the size is too small, omitting procedure for extra experiences is frequently executed. Therefore, these experiences cannot be well-reused.

As mentioned above, there are many important concerns deriving from Dyna-Q architecture and experience replay methods. One is how to record these collected experiences in the virtual model which is used for reinforcement learning, especially for deep reinforcement learning to accelerate the learning process [21]-[23]. Another concern is that these methods in planning always reuse the experiences which are collected by the agent in the past. This paper presents an adaptive tree structure with experience replay, called ERTS-Q for model learning in continuous state space applications. In RL applications, the ERTS-Q method does not determine the resolution of the tree structure in advance, the tree structure can adaptively decide the resolution. In the beginning, an agent handles a tree structure represented by a root node corresponding to the entire state space. In the process of learning, the agent can adaptively split the state space. Internal nodes, corresponding to subspaces, are created during the process of model learning. Statistical information based on the variation of the current state and the next state is calculated and is recorded in each tree nodes. For a large continuous state space, some states do not be visited by the agent. Based on the tree structure, it is adaptive to discretize the continuous state space, so, the agent creates the virtual experiences which are not visited by the agent before. Therefore, in planning, the agent can simulate many virtual experiences based on the variation between current states and the next states. Based on the tree model, many virtual experiences which have been visited and have not been visited before are created for extra updating the action-value function. Therefore, the sampling efficiency can be achieved, and the process of learning is accelerated.

The remainder of this paper is organized as follows. Section II introduces reinforcement learning and Dyna architecture. The proposed ERTS-Q approach based on tree structure is described in Section III. Section IV discusses the simulated results to show the validity of the proposed method. The final section gives conclusions.

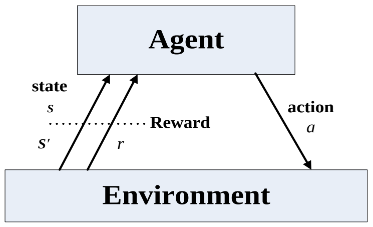

Fig. 1. Reinforcement learning architecture.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) agent interacts with the environment without any knowledge about it. A trial-and-error process is used to collect interactive experience for policy learning [24]. After many learning iterations, the collecting experiences are used to train the policy. The agent uses its own policy to select sequences of actions to achieve the goal. In each learning iteration, as shown in Fig. 1, the agent obtains a current continuous state which will be changed to a discretized state, $\boldsymbol{s} \in \boldsymbol{S}$, which is state space and uses its own policy to select an action, $a \in \boldsymbol{A}$, which is action space. When an action a is taken, the agent will obtain a next continuous state which is discretized as the next state, $\boldsymbol{s}'$, and receive a reward, r, from the environment. The collecting experiences, $(\boldsymbol{s}, a, \boldsymbol{s}', r)$, are used to update the action-value function.

2.2 Dyna Architecture

Dyna architecture, which combines policy learning and model learning, is an extension of reinforcement learning (RL) [13][15]. Many policy learning methods such as Q-Learning, SARSA and Actor-critic are used for learning [24]. Model learning includes two functions, transition function, and reward function, which are viewed as function approximators [13][15]. In [13][24], the model handles two functions, transition function $\boldsymbol{P}(\boldsymbol{s}' \mid \boldsymbol{s}, a)$ and reward function $\boldsymbol{R}(\boldsymbol{s}, a)$. $\boldsymbol{P}(\boldsymbol{s}' \mid \boldsymbol{s}, a)$ is from any state $\boldsymbol{s}$ and action a, the transition of each possible next state, $\boldsymbol{s}'$, is

$$\boldsymbol{P}(\boldsymbol{s}' \mid \boldsymbol{s}, a) = \{ \boldsymbol{s}_{t+1} = \boldsymbol{s}' \mid \boldsymbol{s}_t = \boldsymbol{s}, a_t = a \} \quad (1)$$

In the same way, the expected reward of executing action, a, in the current state, $\boldsymbol{s}$, transiting to next state, $\boldsymbol{s}'$, is

Algorithm 1: The Dyna Algorithm

```
S : action space
A : action space
Initial Q(s, a) and Model(s, a) for all states s ∈ S and actions a ∈ A
Repeat:
1.    $\boldsymbol{s}_t \leftarrow$ current state, $a_t \leftarrow \varepsilon$-greedy($\boldsymbol{s}_t, Q(\boldsymbol{s}_t, a_t)$)
2.    Execute action $a_t$; observe next state, $\boldsymbol{s}_{t+1}$, and reward, $r_{t+1}$
3.    Update Q-value
          $Q(\boldsymbol{s}_t, a_t) \leftarrow (1-\beta)Q(\boldsymbol{s}_t, a_t) + \beta(r + \gamma \max_{a_{t+1}} Q(\boldsymbol{s}_{t+1}, a_{t+1}))$
4.    Model($\boldsymbol{s}_t, a_t$) $\leftarrow$ ($\boldsymbol{s}_{t+1}, r_{t+1}$)
Repeat T times:
5.    $\boldsymbol{s} \leftarrow$ random previously perceived state
6.    $a \leftarrow$ random action previously taken in state $\boldsymbol{s}$
7.    ($\boldsymbol{s}', r$) $\leftarrow$ Model($\boldsymbol{s}, a$)
8.    Update Q-value
          $Q(\boldsymbol{s}, a) \leftarrow (1-\beta)Q(\boldsymbol{s}, a) + \beta(r + \gamma max_{a' \in A} Q(\boldsymbol{s}', a'))$
```

$$\boldsymbol{R}(\boldsymbol{s}, a) = \{ r_{t+1} \mid \boldsymbol{s}_t = \boldsymbol{s}, a_t = a \} \quad (2)$$

It contains two kinds of phases—phase of model learning and phase of planning. In the phase of model learning, a Dyna agent uses real experiences collected from the environment, to build a model approximating the environment and to update its action-value function. In the phase of planning, when the model approximates the environment, the model can produce many simulated experiences and use them for extra learning. The algorithm of Dyna architecture is shown in **Algorithm** 1. In each learning iteration, the agent first interacts with the environment directly to collect experience which consists of $\boldsymbol{s}$, a, $\boldsymbol{s}'$, and r. Then, the collected experience is used to update the action-value function and build a model which is similar to the real environment. In the phase of planning, the agent generates simulated experience from the model for extra learning at steps 5, 6 and 7. Therefore, the Dyna agent acquires extra simulated experience to speed up the learning.

3 Experience Replay based on Tree Structure with Q-Learning

The main contribution of this paper is an algorithm, called experience replay method based on a tree structure with Q-Learning (ERTS-Q) which is used to achieve the high sample efficiency by using an approximated model. Model-based RL including transition function and reward function is extended from reinforcement learning. However, how to build a model for an environmental approximator is not within the scope of model-based RL [14]. In this paper, the ERTS-Q algorithms include three parts: 1) Q-Learning is used for policy learning; 2) a tree-structure, called tree-model is used for model learning; 3) an experience replay method based on an action selection function is used for simulating virtual experiences. The ERTS-Q method, with no prior knowledge of the environment, the agent must collect experiences from the environment that is for executing policy learning and model learning. The detailed procedure of ERTS-Q algorithm is specified as follows.

3.1 Policy Learning based on Q-Learning

In the process of learning, the agent perceives environmental information in the form of a continuous state, $\boldsymbol{i}_t$, which is discretized as a discrete state, $\boldsymbol{s}_t$. Then, it selects an action, a, based on its policy (ε-Greedy). After executing the action, a_t, the agent observes the next transiting state, $\boldsymbol{i}_{t+1}$ and receives a reward, r_{t+1}, sequentially. The continuous state, $\boldsymbol{i}_{t+1}$, is discretized as the state $\boldsymbol{s}_{t+1}$. In the process of learning, this real experience—($\boldsymbol{s}_t$, a, $\boldsymbol{s}_{t+1}$, r_{t+1}) in a discrete form—is used to adjust the Q-functions. In the meantime, the agent records the continuous state, $\boldsymbol{i}_t$, into the set $\boldsymbol{I}_R$. This real experience—($\boldsymbol{i}_t$, a, $\boldsymbol{i}_{t+1}$, r_{t+1}) in a continuous form—is used for model learning. A model-free algorithm, Q-Learning, learns an action-value function (Q-function) according to a state $\boldsymbol{s}_t$, an action, a_t, and a Q-value ($Q(\boldsymbol{s}_t, a_t)$) [25]. Given a state, $\boldsymbol{s}_t$, and an action, a_t, the Q-value of the state-action pair, ($\boldsymbol{s}_t$, a_t), is updated by (3).

Algorithm 2: The ERTS-Q algorithm for policy learning

```
A : action space
Initialize I_R : recorded continuous states
Policy learning:
1.  $\boldsymbol{i}_t \leftarrow$ current continuous state
2.  $\boldsymbol{s}_t \leftarrow$ discrete($\boldsymbol{i}_t$), $a_t \leftarrow$ $\varepsilon$-greedy($\boldsymbol{s}_t$, $Q$)
3.  Execute action $a_t$; observe next continuous state,
    $\boldsymbol{i}_{t+1}$, and reward, $r_{t+1}$
4.  if $\boldsymbol{i}_t \notin \boldsymbol{I}_R$ then $\boldsymbol{I}_R \leftarrow \{ \boldsymbol{i}_t \}$
5.    $\boldsymbol{s}_{t+1} \leftarrow$ discrete($\boldsymbol{i}_{t+1}$)
6.  Update $Q$-value
        $Q(\boldsymbol{s}_t, a_t) \leftarrow (1-\beta)Q(\boldsymbol{s}_t, a_t) + \beta(r + \gamma \max_{a_{t+1}} Q(\boldsymbol{s}_{t+1}, a_{t+1}))$
Model learning:
7.  Update-TreeModel ($\boldsymbol{i}_t$, $a_t$, $\boldsymbol{i}_{t+1}$, $r_{t+1}$)
Experience replay based on action selection function:
8.  Experience-replay()
```

$$Q(\boldsymbol{s}_t, a_t) \leftarrow \\ (1-\beta)Q(\boldsymbol{s}_t, a_t) + \beta(r + \gamma \max_{a_{t+1}} Q(\boldsymbol{s}_{t+1}, a_{t+1})) \quad (3)$$

where $\beta\,(0 \le \beta \le 1)$ is the learning rate, and $\gamma(0 \le \gamma \le 1)$ is the discount factor. Q-Learning (tabular) has been proven to converge with a probability one [25]. The ERTS-Q algorithm for policy learning is shown in **Algorithm** 2. In step 7, the real experience in a continuous form is used for model learning to approximate the environment. In step 8, when the agent stays in the phase of planning, it uses experience replay method to produce virtual experiences for extra learning. The procedure will decrease the cost for the agent to explore the environment.

3.2 Model Learning based on Tree Structure

In the original model-based method, agents input state-action pairs ($\boldsymbol{s}_t$, a_t) to the model which will predict the next state and reward. For a huge continuous state space, memory space needs to record all environmental information that would be a challenge. To use memory space efficiently, this paper proposes the concept of variation of continuous state stored in different tree nodes. Fig. 3 is an example of variations of continuous states between two continuous states in a continuous state space. An agent stays in different continuous states, $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, and $\boldsymbol{i}_3$, selects the same action, a, transits to the next continuous states, $\boldsymbol{i}_1'$, $\boldsymbol{i}_2'$ and $\boldsymbol{i}_3'$, and receives rewards, r_1, r_2, and r_3, respectively. The variations of continuous state, $\Delta\boldsymbol{i}_a$, between the continuous states, $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, and $\boldsymbol{i}_3$, and the next continuous states, $\boldsymbol{i}_1'$, $\boldsymbol{i}_2'$ and $\boldsymbol{i}_3'$, are calculated as $\Delta\boldsymbol{i}_{a,1}$ = (1, 1), $\Delta\boldsymbol{i}_{a,2}$ = (1, 1), and $\Delta\boldsymbol{i}_{a,3}$ = (1, 0), respectively. According to the example shown in Fig. 2, the variations of continuous state, $\Delta\boldsymbol{i}_{a,1}$ and $\Delta\boldsymbol{i}_{a,2}$ which have the same variations, and $\Delta\boldsymbol{i}_{a,3}$ is not. A tree structure mapping the state space to leaf nodes is shown in Fig. 3. In this structure, the continuous states with the same variations are aggregated into the same leaf nodes(Ranges). Otherwise, the different variations are aggregated into the different nodes (Ranges).

- ***Collecting samples (experiences)***

In the proposed tree-model, it only takes one root node corresponding to the entire state space, initially. When a continuous state-action pair, ($\boldsymbol{i}_t$, a_t, $\boldsymbol{i}_{t+1}$, r_{t+1}), is collected, it is fed into the tree-model and is dropped to a corresponding leaf node. The variation between this continuous state, $\boldsymbol{i}_t$, and the next continuous state, $\boldsymbol{i}_{t+1}$, is calculated as (4).

$$\Delta\boldsymbol{i}_a \leftarrow \boldsymbol{i}_{t+1} - \boldsymbol{i}_t \tag{4}$$

where a denotes an action, $\boldsymbol{i}_t = [i_1, i_2,\ldots, i_d,\ldots, i_n]$ and $\boldsymbol{i}_{t+1} = [i'_1, i'_2,\ldots, i'_d,\ldots, i'_n]$ represent this continuous state and the next continuous state, $\Delta\boldsymbol{i}_a = [\Delta i_1, \Delta i_2,\ldots, \Delta i_d,\ldots, \Delta i_n]$, $d = 1, 2,\ldots, n$, represents the variation of the continuous state-action pair, n is dimension of the continuous state.

After calculating the variation, in this node, the number of the samples, N_a, is increased. If the number of collecting samples is smaller than the pre-defined threshold, N_{th}, the agent needs to collect samples continuously. Otherwise, the agent must decide to split the tree-model or not. If the agent decides to split the tree-model, the average of the variations of the successive states, $\boldsymbol{\mu}_a^{\boldsymbol{i}}$, and the average of rewards, μ_a^r, are calculated. The $\boldsymbol{\mu}_a^{\boldsymbol{i}}$ and μ_a^r are calculated by equations (5), respectively.

$$\begin{aligned}\overline{\Delta\iota_d} &\leftarrow \left(\sum_{h=1}^{N_a}\left(i'_{h,d} - i_{h,d}\right)\right)/N_a \\ \mu_a^r &\leftarrow \left(\sum_{h=1}^{N_a}\left(r_{h,d}\right)\right)/N_a\end{aligned} \tag{5}$$

where N_a is the number of samples, $d = 1, 2,\ldots, n$, $\boldsymbol{\mu}_a^{\boldsymbol{i}} = [\overline{\Delta\iota_1}, \overline{\Delta\iota_2},\ldots, \overline{\Delta\iota_d},\ldots, \overline{\Delta\iota_n}]$, is the average vectors formed according to the average variations of the continuous states in each dimension, and μ_a^r is the mean value that is calculated by rewards.

After obtaining the average of the variation of the continuous states, the variance of the variation of the continuous states and the variance of the rewards are calculated as follows.

$$\begin{aligned}(\overline{\sigma_d})^2 &\leftarrow \left(\left(\sum_{h=1}^{N_a}\left(i'_{h,d} - i_{h,d}\right)\right)/N_a\right) - (\overline{\Delta\iota_d})^2 \\ (\sigma_a^r)^2 &\leftarrow \left(\left(\sum_{h=1}^{N_a}\left(r_{h,d}\right)\right)/N_a\right) - (\mu_a^r)^2\end{aligned} \tag{6}$$

where $(\boldsymbol{\sigma}_a^{\boldsymbol{i}})^2 = [(\overline{\sigma_1})^2, (\overline{\sigma_2})^2,\ldots, (\overline{\sigma_d})^2,\ldots, (\overline{\sigma_n})^2]$, represents variance vectors that are formed according to the variances for the variations in the continuous states in each dimension, and $(\sigma_a^r)^2$ is a variance value, which is calculated by rewards, $d = 1, 2,\ldots, n$.

In the process of learning, the agent will collect more and more experiences and feed into the tree-model for approximating the environment. In Fig. 3, the range corresponding to the subspace of state space includes the same average variation of the continuous states, $\boldsymbol{\mu}_a^{\boldsymbol{i}}$, and rewards, μ_a^r. The variances—$(\boldsymbol{\sigma}_a^{\boldsymbol{i}})^2$ and $(\sigma_a^r)^2$, recorded in the leaf nodes are used to determine whether the leaf nodes should be split. The elements of the variance vectors, $(\overline{\sigma_d})^2 \in (\boldsymbol{\sigma}_a^{\boldsymbol{i}})^2$, where $d = 1, 2,\ldots, n$, are used to guide the dimension d, which cannot be well-represented by the elements of the average vectors, $\overline{\Delta\iota_d}$. Similarly, the variance value, $(\sigma_a^r)^2$, is used to guide the dimension of the reward, which cannot be well-represented by the average value, μ_a^r. If the elements of the variance vectors are smaller than the predefined threshold, σ_{sth}^2 or if the variance value is smaller than the predefined threshold, σ_{rth}^2, and the leaf node is not split. However, large variances signify that the estimations of $\boldsymbol{\mu}_a^{\boldsymbol{i}}$ and μ_a^r are not effective, and the leaf node needs to be split. Besides, if samples are sufficiently large in terms of modeling accuracy, the leaf node may need to split. Due to the diversity of the samples in each dimension, the normalization process must be taken prior to the classification process. The conditions to split a leaf node are listed as follows.

- Condition 1: The number of samples N_a is larger than the predefined threshold N_{th}.
- Condition 2: Any elements in the variance vector, $(\boldsymbol{\sigma}_a^{\boldsymbol{i}})^2$, is larger than the predefine threshold, σ_{sth}^2.
- Condition 3: The variance value of the reward, $(\sigma_a^r)^2$, is larger than the predefined threshold value σ_{rth}^2.

● *Samples Classification*

When samples stored in a leaf node satisfy the condition 1 and either condition 2 or condition 3, the leaf node is split; otherwise, the leaf node collects samples continuously. When all conditions are satisfied, the samples stored in the leaf node are classified according to the average variation of the continuous states in dimension, $d_{\max}$, with maximum variance or the average rewards when the leaf node needs to be split. An example of classifying samples into two classifications in dimension, $d_{\max}$, is depicted in Fig. 4. Samples for classification use the average variation of the continuous state in the dimension, $d_{\max}$, with maximum variance. The variations of samples in classification 1, represented by blue solid circle in Fig. 4, are smaller than or equal to the average variation of continuous states in dimension, $d_{\max}$. The remaining samples, represented by red solid triangle in Fig. 4, are classified into classification 2. The proposed classified samples approach is much simpler and time-saving than our previous work, which applied a simple Adaptive Resonance Theory (ART) method [26]. The neural network methods [26] spent much more computational time to calculate split points, but only slightly improve per-

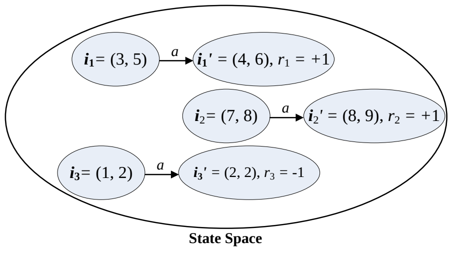
$\boldsymbol{i}_1$= (3, 5)
a
$\boldsymbol{i}_1'$ = (4, 6), r_1 = +1
$\boldsymbol{i}_2$= (7, 8)
a
$\boldsymbol{i}_2'$ = (8, 9), r_2 = +1
$\boldsymbol{i}_3$= (1, 2)
a
$\boldsymbol{i}_3'$ = (2, 2), r_3 = -1
State Space

Fig. 2. An example for calculating the variations of continuous state.

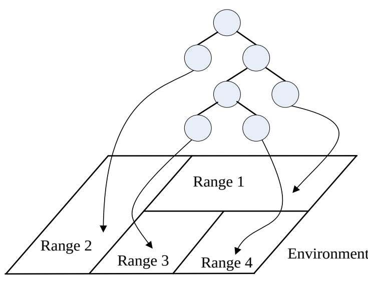

Fig. 3. A tree structure for the model learning and experience replay.

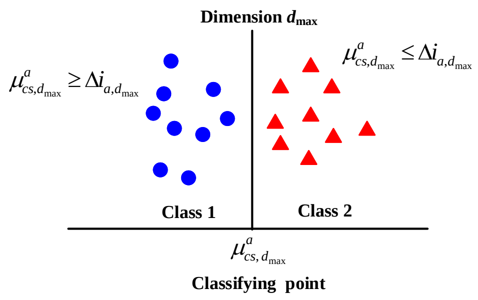
Dimension $d_{\max}$
$\mu^{a}_{cs,d_{\max}} \geq \Delta i_{a,d_{\max}}$
$\mu^{a}_{cs,d_{\max}} \leq \Delta i_{a,d_{\max}}$
Class 1
Class 2
$\mu^{a}_{cs,d_{\max}}$
Classifying point

Fig. 4. An example of classifying samples into two classes (leaf nodes).

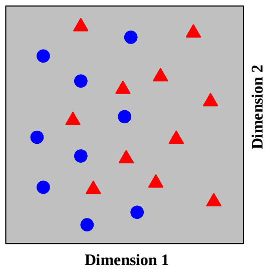

Fig. 5. All classified samples are mapped into the subspace.

formance in classification.

- ***Selecting the splitting dimension***

After classifying the samples, the tree-model needs to determine which dimension of the continuous state space should be split. As shown in the example in Fig. 5, the classified samples are mapped to the sub-state space and there are two sample groups, $G = [g_1, g_2]$. The samples in group 1 which have smaller variations of the continuous states in dimension, $d_{\max}$ are represented by blue solid circles. The samples which have larger variations of continuous states are represented by solid red triangles. The elements in the same group are regarded as statistical samples and written as $X^{a}_{g_{l,d}} \sim \left(\mu^{a}_{g_{l,d}}, \left(\sigma^{a}_{g_{l,d}}\right)^2\right)$, where $\mu^{a}_{g_{l,d}}$ and $\left(\sigma^{a}_{g_{l,d}}\right)^2$ represent the average variance of dimension d of group g_l, l = 1, 2 and d = 1, 2, …., n. Because the samples include n dimensions, there are n splitting values that need to be estimated. The example, shown in Fig 6, has two dimensions, and thus, has two splitting values to be evaluated. The splitting value of dimension d is the average of mean (*AOM*) values of all groups in dimension d, as described in (7), where $|G|$ denotes the number of groups. Equation (8) defines the probability of group l, where $|g_l|$ denotes the number of samples in group l. The entropy function of samples is described in (9), and it is furthermore used to evaluate the split value of each dimension.

$$splitting\_value_d \leftarrow \sum_{l=1}^{|G|} \left(\mu^{a}_{g_l,d}\right) \Big/ |G| \tag{7}$$

$$P(g_l) \leftarrow |g_l| \Big/ \sum_{l=1}^{|G|} |g_l| \tag{8}$$

$$purity[P(G)] \leftarrow \sum_{l=1}^{|G|} \left(P(g_l)\log(P(g_l))\right) \tag{9}$$

When the values of the entropy function are calculated, the splitting values could be chosen by using equation (10), which selects the splitting value with the maximum entropy value.

$$splitting\_value \leftarrow \arg\max_{sv} \left\{ purity_d\left[P(G_{sv,left})\right] + purity_d\left[P(G_{sv,right})\right] \right\} \tag{10}$$

where $G_{sv,left}$ and $G_{sv,right}$ denote the samples of the group sets that belong to the left and right child nodes, respectively, after splitting at *splitting_value*, *sv*. From Fig. 6, the splitting value 2 can obtain the maximum entropy value using the entropy function listed in equation (10). The collected samples are received following the trajectory of the agent, so the tree-model is split along the same dimension in the same local area (group).

Fig. 7 shows an improper splitting value in a sub-state space (group/node), wherein the solid circles and solid triangles represent the collected samples. The splitting value 2 obtains a larger entropy value than splitting value 1, the node will select the splitting value 2 as the splitting point. And then, the upper node is split at splitting point 2 in the next step. By using equation (10) to choose the splitting value, it produces many threadlike leaf nodes. In other words, using the equation (10) for choosing a splitting value cannot accurately reflect the situations of the sampling distribution. The range factor of the dimension is taken into consideration for determining which dimension should be chosen for splitting. The factor can be used to avoid redundancy such as the threadlike leaf nodes. Therefore, equation (10) is refined as (11). The *range*$_d$, defined in (11), denotes the value range of dimension d in a particular leaf node. The *range*$_{d,L}$ and *range*$_{d,U}$ are the lower bound and upper bound of data value in dimension d, respectively.

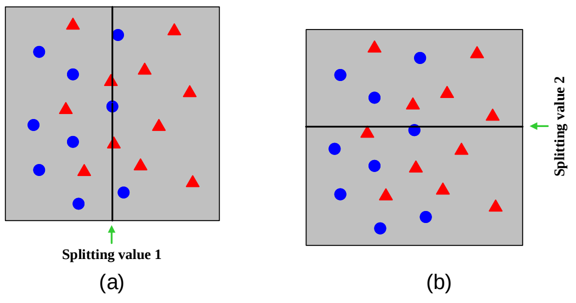

Fig. 6. An example of sample distribution for different splitting values (a). Splitting value 1 (b). Splitting value 2.

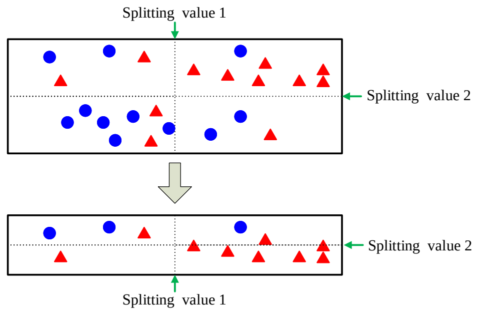

Fig. 7. An example of choosing improper splitting value according to equation (10).

Algorithm 3: The Algorithm of Model Learning for Tree-Model

```
N ← size of continuous states
A : set of actions
Sv : set of splitting values
Model learning:
1.  for d ← 1 to n do
2.    $\Delta \boldsymbol{i}_a \leftarrow \boldsymbol{i}_{t+1} - \boldsymbol{i}_t$
3.  end for
4.  Calculate_Average_Variance()  // by equations (5)(6)
5.  SP ← False
6.  if $N_a > N_{th}$ then
7.    for d ← 1 to n do
8.      if $(\boldsymbol{\sigma}_a^{\boldsymbol{i}})^2 > \sigma_{sth}^2$ then
9.        SP ← True
10.     else if $(\sigma_a^r)^2 > \sigma_{rth}^2$ then
11.       SP ← True
12.     end if
13.   end for
14. end if
15. if SP is True then // split the tree-model
16.   Classify_Samples(); //illustrated in Fig. 4
      /* to compute splitting values for dimensions by equation (7) */
17. Sv ← Calculate_SplittingValue()
      /* choose the split value by equation (11)
18. spiltting_value ← Calculate_Weighted_PurityFunction(Sv)
19. Split_the_Tree-Model(spiltting_value)
20. end if
```

Algorithm 4: The ERTS-Q algorithm in planning

```
A : action space
Initialize $\boldsymbol{I}_R$ : recorded continuous states
Experience replay based on action selection function:
1.  Repeat T time
2.    $\boldsymbol{i}_t$ ← randomly observed continuous states from $\boldsymbol{I}_R$
3.    $\tilde{a}$ ← select an action according to action selection functions
4.    $(\boldsymbol{\mu}_a^{\boldsymbol{i}}, \bar{r}_a)$ ← ExperienceReplay($\boldsymbol{i}_t$, $\tilde{a}$)
5.    $\boldsymbol{i}_{t+1} \leftarrow \boldsymbol{i}_t + \boldsymbol{\mu}_a^{\boldsymbol{i}}$
6.    Compute-ActionValues($\boldsymbol{i}_t$, $\tilde{a}$, $\boldsymbol{i}_{t+1}$, $\bar{r}_a$)
```

$$splitting\_value \leftarrow \arg\max_{sv}\left\{purity_d\left[P\left(G_{sv,left}\right)\right] + purity_d\left[P\left(G_{sv,right}\right)\right] / range_d\right\} \quad (11)$$

$$range_d = range_{d,U} - range_{d,L}$$

The range factor is used to reflect the distribution. The splitting value obtains a larger entropy value and the range factor is small. In this situation, the entropy value will be lowered to reflect the distribution. In contrast, the splitting value obtains a smaller entropy value and the range factor is large. The entropy value will be heightened. After selecting the splitting value following the rules, two new children nodes are produced and inherit the information from the parent node. The detailed tree-model for model learning is shown in **Algorithm** 3.

3.3 StructureExperience Replay based on An Action Selection Function

● *An Action Selection Function*

An agent uses the model to produce the simulated experience for extra policy learning that is called planning. The detailed step of ERTS-Q in planning is shown in **Algorithm** 4. In planning, the agent replaces the environment with a tree-model to achieve the experience replay. The process of experience replay is that it retrieves a continuous state, i_t, from the set, $\boldsymbol{I}_R$. An action selection function at step 3 is used to decide an action, $\tilde{a}$. The objective of planning is to lower the temporal error, defined in (12), as soon as possible.

In the Dyna framework, the agent obtains interactive experience from the environment and simultaneously records the experience. The agent in planning uniformly picks up state-action pairs stored in the past, to refine the Q-value. The Dyna framework randomly picks up the state-action pairs stored in the model for value iteration [13][15]. In the proposed planning method, the agent picks up the number of N continuous states, $\boldsymbol{i}_t$, from the set, $\boldsymbol{I}_R$. Two action selection methods are introduced to decide an action, $\tilde{a}$, in the continuous states, $\boldsymbol{i}_t$. The first one is a random action function. Because the action space is finite and known, the random action function could uniformly select an action from the action space [27]. The second one is action selection function. Whereas, the action selection function, defined in (13), selects an action according to the current information.

$$\tilde{a} \leftarrow \arg\max_{a_t \in A}\left[(1-\lambda)\Delta Q(s_t, a_t) + \lambda\sqrt{\log\left(C_{Node(\boldsymbol{i}_t)}\right) / C_{Node(\boldsymbol{i}_t, a_t)}}\right] \quad (13)$$

where $\lambda \in [0,\ 0.5)$ is a predefined constant. $C_{Node(\boldsymbol{i}_t)}$ is the total number of planning times for a continuous state, $\boldsymbol{i}_t$, on certain leaf nodes. $C_{Node(\boldsymbol{i}_t, a_t)}$ is the number of planning times in the leaf node where the continuous state, $\boldsymbol{i}_t$, resides with the action, a_t, which belongs to the action set, $\boldsymbol{A}$. If $C_{Node(\boldsymbol{i}_t, a_t)} = 0$ is replaced by a constant $v \in [0,\ 1)$, then the equation directs the action selection to focus more on the larger error, ΔQ, at the initial stage. If ΔQ approaches zero that the temporal error is stable, the second term on the right side represents a type of virtual exploration increases the preference for the less selected actions.

● *Experience Replay for Simulated experiences*

When the agent gets the simulated continuous state-action ($\boldsymbol{i}_t$, $\tilde{a}$), it inputs it into the tree-model to generate simulated experiences. The tree-model handles two functions, variation function of the continuous state, $\boldsymbol{P}(\Delta\boldsymbol{i} \mid \boldsymbol{i}_t, \tilde{a})$, and reward function, $\boldsymbol{R}(\boldsymbol{i}_t, \tilde{a})$. $P(\Delta\boldsymbol{i} \mid \boldsymbol{i}_t, \tilde{a})$is from any continuous state $\boldsymbol{i}_t$ and action $\tilde{a}$, which selects from equation (13). The average variation of the continuous state, $\boldsymbol{\mu}_a^{\boldsymbol{i}}$, is defined as follows.

$$\boldsymbol{P}(\boldsymbol{\mu}_a^{\boldsymbol{i}} \mid \boldsymbol{i}, a) = \{\boldsymbol{\mu}_a^{\boldsymbol{i}} \mid \boldsymbol{i}_t = \boldsymbol{i},\ \tilde{a} = a\} \quad (14)$$

In the same way, the expected reward of executing action, a, in the current state, $\boldsymbol{i}$, transiting to next state, $\boldsymbol{i}'$, is

$$\boldsymbol{R}(\boldsymbol{i}, a) = \{\ r_{t+1} \mid \boldsymbol{i}_t = \boldsymbol{i},\ \tilde{a} = a\ \} \quad (15)$$

According to Chebyshev's theorem [24], the probability of the variation function of the continuous state in each dimension is formulated as

$$\mathbf{P}(|X - \overline{\Delta\iota_d}| \le k\overline{\sigma_d}) \ge \left(1 - 1/k^2\right) \quad (16)$$

where X represents the data, k is a constant, $\overline{\Delta\iota_d} \in \boldsymbol{\mu}_a^{\boldsymbol{i}}$, $\overline{\sigma_d} \in \sigma_a^{\boldsymbol{i}}$, the probability is decided according to Chebyshev's theorem [27], which provides a lower bound for the probability that data appears within the intervals, $[\overline{\Delta\iota_d} - k\overline{\sigma_d}, \overline{\Delta\iota_d} + k\overline{\sigma_d}]$. If the data is normally distributed, the probability is about 95%.

The tree-model uses the continuous states and actions as an index to search the leaf nodes which calculates the probability and obtains the means of the variations in the continuous state, $\overline{\Delta\iota_d}$ in each dimension and the reward $\bar{r}_a$. The next continuous state, $\boldsymbol{i}_{t+1}$ is obtained by using equation (17).

$$\boldsymbol{i}_{t+1} \leftarrow \boldsymbol{i}_t + \boldsymbol{\mu}_a^{\boldsymbol{i}} \tag{17}$$

Various numbers of T continuous state-action pairs are produced for experience replay. The planning phase is executed by using the same Q-learning for the simulated experiences as that used for learning. Instead of generating a set of candidate pairs from which a prevailing pair is then chosen, which consumes much more computational time, the proposed tree-model deterministically predicts a single pair of transiting states and rewards.

4 Simulations

In this section, two simulations—a mountain car and a mobile robot in a maze—are used to validate the proposed methods. In both cases, a robot explores its environment to learn the policy and to approximate the environmental model. When the environmental model is constructed, the model produces virtual experiences to accelerate learning. To verify the performance of the proposed approach, ERTS-Q method is compared with the Q-learning [22], Q-learning with experience replay based on time and reward [18], the Dyna-Q agent [13][17], and tree-based Dyna-Q agent [27].

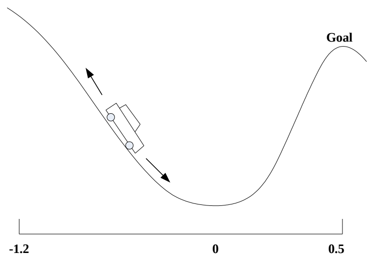

Fig. 8. The example of mountain car.

TABLE I
PARAMETERS OF THE MOUNTAIN CAR SIMULATION

Parameter	Value
Learning rate β	0.7
Discount rate γ	0.8
Exploring rate ε	5%
Planning Number(N)	5
N_{th}	10
σ_{sth}^2	0.032411465
σ_{rth}^2	0.01
λ	0.3

TABLE II
A COMPARISON OF THE SPLITTING METHODS USED FOR THE MOUNTAIN CAR SIMULATION

Splitting methods	Averaged Number of Leaf Nodes	Episodes
Average of mean(AOM)	69.725	14
Mean of range (MOR)	115.925	14
Uniform splitting of samples (USS)	110.8	15

4.1 Mountain car

In this simulation, a car without enough power is placed in a valley, as shown in Fig. 8. To reach the hill, the car must move to the left side to save more potential energy. The car accumulates more energy to climb the hill on the right side. The learning methods try to make the car reach the goal efficiently. The dynamic equation is listed in (18).

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_{t+1} \\ \dot{x}_{t+1} &= \dot{x}_t + 0.001a_t - 0.0025\cos(3x_t) \end{aligned} \tag{18}$$

where x_t is the position of the car at time t and $\dot{x}_{t+1}$ is the velocity of the car.

In this simulation, the position and the velocity of the car form the state space. The position dimension is from -1.2 to 0.5 and the velocity dimension is from -0.07 to 0.07. The action space is composed of three actions, throttle forward, throttle reverse and zero throttles. When the car touches the goal, it receives a positive reward (1). Otherwise, it obtains a negative reward (-1). There are 40 rounds for the mountain car simulation. Each round has 200 episodes and 2000 steps per episode. In each episode, if the number of steps is greater than 2000 or if the car climbs the right hill, the episode is terminated. The initial position and the initial velocity of the car are respectively set at -0.5 and 0 for each episode. The settings of simulated parameters for mountain car task are listed in Table I.

The methods to determine the splitting value are first compared. Equation (19) uses the mean of the ranged intervals (MOR) to compute the split value for each dimension. The uniform splitting of samples (USS) [29] is defined in (20).

$$splitting\_value_d \leftarrow \left(range_{d,L} + range_{j,U}\right)/2.0 \tag{19}$$

$$\begin{aligned} &splitting\_value_{d,z} \leftarrow \\ &\quad spl_{d,L} + \left(spl_{d,U} - spl_{d,L}\right) \times z / \left(N_{split} + 1\right) \end{aligned} \tag{20}$$

where $range_{d,L}$ and $range_{d,U}$ respectively denote the lower and upper bounds of the range in dimension d. z =1, 2, …, N_{split} in (20) is the z^{th} splitting value of a leaf node in dimension d, where N_{split} is the number of split intervals, and it is set to 3 for comparison. $spl_{d,L}$ and $spl_{d,U}$ respectively indicate the lower and upper bounds of the variables in dimension d.

Table II lists the results of comparisons— the number of leaf node and episode, for AOM, defined in (9), MOR, and USS splitting value methods. The parameters settings

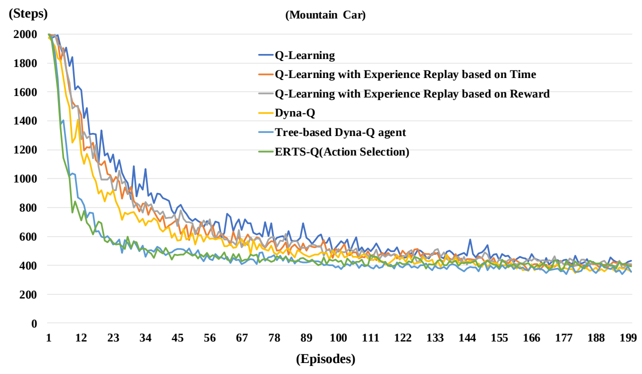

Fig. 9. The simulation results of the mountain car.

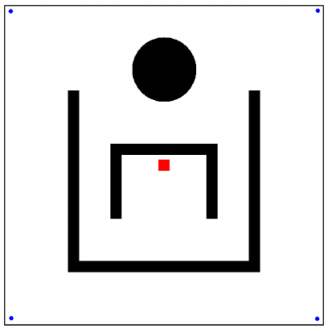

Fig. 10. The example of mobile robot in a maze.

are the same, shown in Table I. The averages of a number of leaf nodes are 69.725, 115.925, and 110.8 for *AOM*, *MOR*, and *USS*, respectively. It is obvious that the *AOM* approach partitions fewer averaged leaf nodes. The three splitting value computing methods use different numbers of leaf nodes to approximate the environment, and they reduce the average number of steps down to 700 steps in similar episodes. According to the simulated results, the performance is not obviously influenced.

Fig. 9 shows the average number of steps required for the car to climb the right-side hill for each episode of 40 rounds training. The six curves represent Q-learning, Q-learning with experience replay based on time and based on reward, a Dyna-Q agent, a tree-based Dyna-Q agent, and the ERTS-Q agent which reduces the average number of steps to 600 steps on the 16th episodes, but 74 episodes are required for Q-learning, 49 and 58 episodes for Q-learning with experience replay based on time and based on reward, 43 episodes for the Dyna-Q agent, and 19 episodes for the tree-based Dyna-Q agent. It is obvious that the proposed method can accelerate learning. Although the proposed method is faster than the Dyna-Q agent with random action in this platform, the difference between these two methods is not obvious.

4.2 Mobile robot in a maze

In Fig. 10, a mobile robot is placed in a maze and tries to approach its goal. Walls surround a 300x300 sized maze. The starting points for the robot are set in the corner of the maze. So, the robot randomly starts in four corners in each episode. The rectangle in the center of the maze is the area of the goal. The U-shaped blocks and the black solid circle are obstacles. For actions, move up, move down, move left and move right, form the action space. The moving distance of each is set as 5.0. The robot touches the area of the goal after taking each action in a state, it will obtain a positive reward (+100). If the robot hits walls or obstacles, it will not change its position and the environment will return a negative reward (-100). In other situation, the robot will receive a small negative reward (-0.00001). In the process of learning, the number of learning steps reaches 5000 or the robot touches the goal, the episode is terminated. There are 500 episodes for each round. All parameters using in this simulation are listed in Table III.

Table IV lists the results of comparisons— the number of leaf node and episode, for *AOM*, defined in (9), *MOR*, and *USS* splitting value methods. The averages of a number of leaf nodes are 441.77, 609.85, and 686.07 for *AOM*, *MOR*, and *USS*, respectively. It is obvious that the *AOM* approach partitions fewer averaged leaf nodes. Although the *MOR* and *USS* methods produce more leaf modes than *AOM*, the three splitting methods reduce the average number of steps to 1000 in a similar number of episodes.

TABLE III
PARAMETERS OF THE MOBILE ROBOT IN A MAZE SIMULATION

Parameter	Value
Learning rate β	0.9
Discount rate γ	0.85
Exploring rate ε	30%
Planning Number(N)	15
N_{th}	15
σ^2_{sth}	0.3
σ^2_{rth}	20.0
λ	0.9

TABLE IV
A COMPARISON OF THE SPLITTING METHODS USED FOR THE MOBILE ROBOT IN A MAZE SIMULATION

Splitting methods	Averaged Number of Leaf Nodes	Episodes
Average of mean(*AOM*)	441.77	23
Mean of range (*MOR*)	609.85	24
Uniform splitting of samples(*USS*)	686.07	24

Fig. 11 shows three kinds of tree model partitions. All the partitions are fine when close to the walls, obstacles, or the goal. However, the variations in the continuous states and the rewards are more unstable when the partitions are close to the walls and obstacles. The variations in the rewards are especially greater when the partitions are close to the goal. Fig. 12 shows the average number of steps required for the robot to reach the goal for each episode with 40 rounds of training. The six curves represent Q-learning, Q-learning with experience replay based on

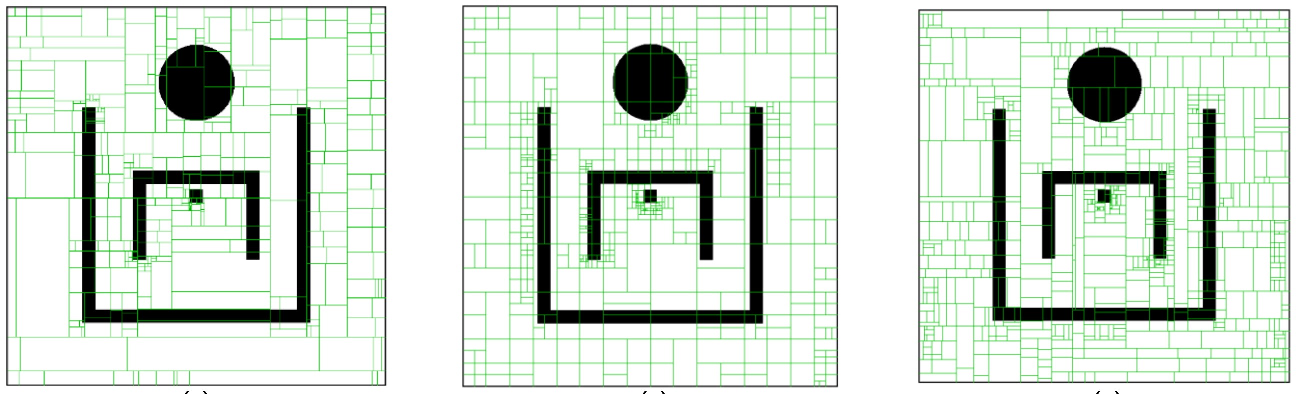

Fig. 11. Model partitioning of the mobile robot in a maze simulation. (a). The *AOM* method (b). The *MOR* method (c). The *USS* method.

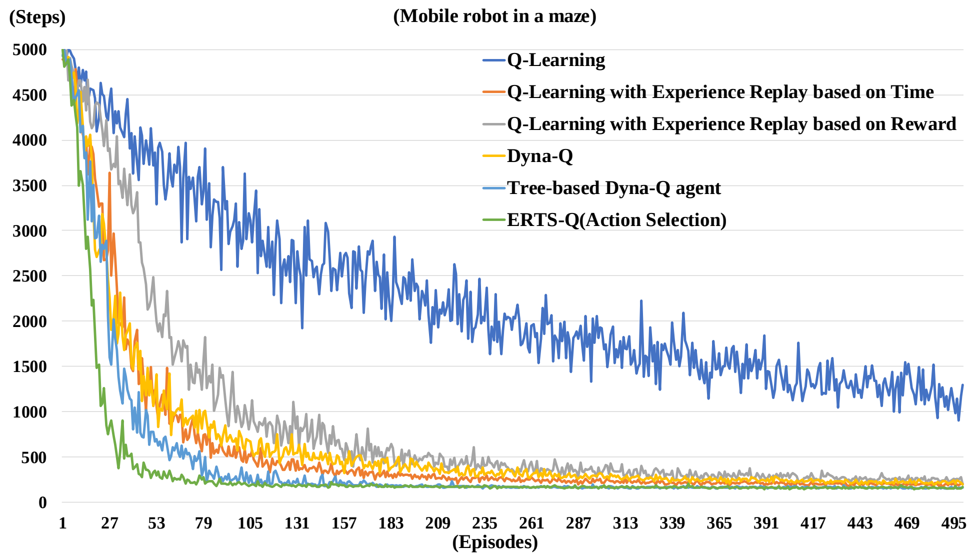

Fig. 12. The simulation results of the mobile robot in a maze.

time and based on reward, a Dyna-Q agent, a tree-based Dyna-Q agent, and the ERTS-Q agent which reduces the average number of steps to 1000 steps on the 25th episodes, but 465 episodes are required for Q-learning, 54 and 84 episodes for Q-learning with experience replay based on time and based on reward, 54 episodes for the Dyna-Q agent, and 40 episodes for the tree-based Dyna-Q agent. It is clear that the ERTS-Q methods can accelerate learning.

5. Conclusions

The original Q-Learning method is hard to achieve sample efficiency that is training a policy to reach a goal in a time constraint. Dyna-Q agent was introduced to improve the original learning methods, but how to establish the model is another problem for model learning. This paper presents ERTS-Q method which improves the efficiency in sampling data in a reinforcement learning problem. The proposed method combines reinforcement learning for policy learning and a tree structure for adaptive model learning. The tree-model is used to approximate the environment by perceiving the transitions between the two successive states and the reward after taking actions. In the initial stages of learning, the tree model is not accurate, so the agent explores the environment as much as possible to collect interactive experience to establish the environmental model. Using the interactive experience, the tree-model becomes more accurate, and then action selection function uses the model to simulate the interactive experience for extra value iterations. An agent using the ERTS-Q method can, therefore, gather simulated experience to update the policy. The splitting methods and the performance of the agent are compared in the simulations. According to the simulations illustrated in Table II and Table IV, none of the three splitting methods clearly influence the learning performance, but there are greater differences in the number of leaf nodes. The proposed ERTS-Q method with tree structure uses fewer leaf nodes to approximate the environment. The simulated results also show that the proposed approach needs fewer steps to achieve the goal than need from Q-Learning or the Dyna-Q agent. Future research will use the concept of prioritized sweeping to further highlight the value of sample efficiency.

REFERENCES

[1] J. Tani, R. Nishimoto, J. Namikawa and, M. Ito, "Codevelopmental Learning Between Human and Humanoid Robot using a Dynamic Neural Network Model," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 38, no. 1, pp. 43-59, Feb. 2008.

[2] S. Wang, W. Chaovalitwongse and R. Babuska, "Machine Learning Algorithms in Bipedal Robot Control," *IEEE Trans. Syst., Man, Cybern., C, Appl. Revi.*, vol. 42, no. 5, pp. 728-743, Sep. 2012.

[3] K. S. Hwang, W. C. Jiang, Y. J. Chen and H. B. Shi, "Motion Segmentation and Balancing for a Biped Robot's Imitation Learning," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 099-1108, Jan. 2017.

[4] S. Doltsinis, P. Ferreira and N. Lohse, "An MDP Model-Based Reinforcement Learning Approach for Production Station Ramp-Up Optimization: Q-Learning Analysis," *IEEE Transactions on Systems, Man, and Cybernetics: System*, vol. 44, no. 9, pp. 1125-1138, 2014.

[5] R. C. Hsu, C.-T. Liu, H.-L. Wang, "A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 2, pp. 181-191, Jun. 2014.

[6] W. T. B. Uther, and M. M. Veloso, "Tree Based Discretization for Continuous State Space Reinforcement Learning," in *Proc. 15th Nat. Conf. Artif. Intell. (AAAI-98)*, Madison, WI, USA, pp. 769-774, 1998.

[7] L. D. Pyeatt and A. E. Howe, "Decision Tree Function Approximation in Reinforcement Learning," *in Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models*, pp. 70-77, 2001.

[8] T. Hester and P. Stone, "Learning and Using Models," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Berlin, Germany: Springer Verlag, 2011.

[9] B. H. Abed-alguni, "Bat Q-Learning Algorithm," *Jordanian Journal of Computers and Information Technology* (*JJCIT*), vol. 3, no. 1, Apr. 2017.

[10] T. Hester, M. Quinlan and P. Stone, "Generalized Model Learning for Reinforcement Learning on a Humanoid Robot," *IEEE Int. Conf. Robo. Autom.* (*ICRA*), Anchorage, Alaska, pp. 2369-2374, 2010.

[11] T. Hester and P. Stone, "Generalized Model Learning for Reinforcement Learning in Factored Domains," in *Proc. 8th Int. Conf. Autonom. Agents Multiagent Syst.*, Budapest, Hungary, pp. 717-724, 2009.

[12] H. H. Viet, P. H. Kyaw, T. Chung, "Simulation-Based Evaluations of Reinforcement Learning Algorithms for Autonomous Mobile Robot Path Planning," in *Proc. Int. Conf. Intell. Robo., Autom., Telecomm. Facil. Appli.*, 2011, pp. 467- 476, 2011.

[13] L. Kuvayev and R.S. Sutton, "Model-based reinforcement learning with an approximate, learned model," in *Proc. 9th Yale Workshop Adapt. Learn. Syst.*, New Haven, CT, USA, pp. 101-105, 1996.

[14] M. Santos, J. A. Martín H., V. López, G. Botella, "Dyna-H: A Heuristic Planning Reinforcement Learning Algorithm Applied to Role-playing Game Strategy Decision Systems," *Know.–Base. Syst.*, vol. 32, pp. 28-36, Aug. 2012.

[15] R. S. Sutton, "Dyna, an integrated architecture for learning, planning and reacting," *ACM SIGART Bull.*, vol. 2, no. 4, pp. 160–163, 1991.

[16] H. H. Viet, S. H. An and T. C, Chung, "Extended Dyna-Q Algorithm for Path Planning of Mobile Robots," *Journal of Measurement Science and Instrumentation*, vol. 2, no. 3, pp. 283-287, Sep. 2011.

[17] H. H. Viet, S. H. An and T. C. Chung, "Dyna-Q-Based Vector Direction for Path Planning Problem of Autonomous Mobile Robots in Unknown Environments," *Advanced Robotics*, vol. 27, no. 3, pp. 159-173, 2013.

[18] M. Pieters, M. A. Wiering, "Q-learning with experience replay in a dynamic environment," *IEEE Symposium Series on Computational Intelligence* (*SSCI*), 2016.

[19] S. Kalyanakrishnan and P. Stone, "Batch Reinforcement Learning in a Complex Domain," *International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 650–657, 2007.

[20] Y. H. Wang, T. H. S. Li, C. J. Lin, "Backward Q-learning: The combination of Sarsa algorithm and Q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184-2193, 2013.

[21] H. van Hasselt, A. Guez, D. Silver, "Deep reinforcement learning with double Q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[22] X. He, K. Wang, T. Miyazaki, H. Huang, Y. Wang, S. Guo, "Green resource allocation based on deep reinforcement learning in content-centric IoT," *IEEE Trans. Emerg. Topics Comput.*, Feb. 2018.

[23] N. Kato et al., "The deep learning vision for heterogeneous network traffic control–Proposal challenges and future perspective", *IEEE Wireless Commun. Mag.*, 2016.

[24] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, U.K.:MIT Press, 1998.

[25] C. Watkins and P. Dayan, "Technical note: Q-Learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279-292, 1992.

[26] K. S. Hwang, H. Y. Lin, Y. P. Hsu and H. H. Yu, "Self-organizing state aggregation for architecture design of Q-learning," *Inf. Sci.*, vol. 181, no. 13, pp. 2813-2822, 2011.

[27] K. S. Hwang, W. C. Jiang and Y. J. Chen, "Tree-based Dyna-Q agent," *IEEE/ASME Int. Conf. Advan. Intell. Mech.(AIM),* pp. 1077-1080, 2012.

[28] B. G. Amidan, T. A. Ferryman, and S. K. Cooley, "Data outlier detection using the Chebyshev theorem," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, USA, 2005, pp. 3814–3819.

[29] K. S. Hwang, Y. J. Chen, W. C. Jiang and, T. W. Yang, "Induced states in a decision tree constructed by Q-learning," *Inf. Sci.*, vol. 213, no. 5, pp. 39-49, Dec. 2012.

Wei-Cheng Jiang received the B.S. degree in Computer Science and information Engineering and M.S. degree in Electro-Optical and Materials Science from National Formosa University, Yunlin, Taiwan, in 2007 and 2009, respectively. He received the Ph.D. degree in Electric Engineering from National Chung Cheng University, Chiayi, Taiwan, in 2013. He was a posdoc with the Electrical Engineering Department at National Sun Yat-sen University, Kaohsiung, Taiwan. He is currently a visiting scholar with Department of Electrical and Systems Engineering at Washington University in St. Louis, Missouri, Amreica. His research interests include machine learning, neural networks, and intelligent control.

Kao-Shing Hwang (M'93_SM'09) is a professor of the Department of Electrical Engineering at National Sun Yat-sen University, an adjunct professor of the Department of Healthcare Administration and Medical Informatic, Kaohsiung Medical University, Taiwan, and a visiting chair professor of the Department of Computer Engineering, Northwestern Polytech. University, Xian, China. He received the M.M.E. and Ph.D. degrees in Electrical and Computer Engineering from Northwestern University, Evanston, IL, U.S.A., in 1989 and 1993, respectively. He had been with National Chung Cheng University in Taiwan from 1993-2011. He was the deputy director of Computer Center (1998-1999), the chairman of the Electrical Engineering Department (2003~2006), and the director of the Opti-mechatronics Institute of the university (2010~2011) in National Chung Cheng University. He has been a member of IEEE since 1993 and a Fellow of the Institution of Engineering and Technology (FIET). His research interest includes methodologies and analysis for various intelligent robot systems, visual servoing, and reinforcement learning for robotic applications.

Jin-Ling Lin received the master and Ph.D. degrees in computer science from the University of Oklahoma, Norman, OK, USA, in 1989 and 1993, respectively. She is currently a professor with the Department of Information Management, Shih Hsin University, Taipei, Taiwan. Her current research interests include machine learning, data science, data mining, information retrieval, intelligent system, intelligent network routing, and path planning in logistics.