Effective Experiences Collection and State Aggregation in Reinforcement Learning

Tao Zhang¹, Ying Liu¹, Yu-Jen Chen², and Kao-Shing Hwang², Senior Member IEEE

¹School of Software, Northwestern Polytechnical University, Shaanxi 710072, China
²Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

Corresponding author: K.-S. Hwang (carlhwang@gmail.com)

ABSTRACT In reinforcement learning systems, learning agents cluster a large number of experiences by identifying similarities in terms of domain knowledge and replace the groups with a representative prototype. Our method addresses two key challenges in reinforcement learning: the difficulty of transferring continuous state domain into the discrete state space, and the need for a good compromise between exploration and exploitation. To tackle the former challenge, the adaptive state aggregation algorithm with a decision tree helps the agent to learn the optimal policy in a continuous state space. For the latter challenge, this study proposes an adaptive state aggregation algorithm, which uses information entropy to evaluate the probability of state-action-dependent exploration and the value for ε is determined using the information entropy instead of manual tuning. Meanwhile, a Tabu search is used to lift the efficiency of exploration. For the planning algorithm, the time required for global exploration depends only on the metric resolution, and not on the size of the state space. The simulation experiments demonstrate the effectiveness of the proposed method, which adaptively partitions the state space into exclusive subspaces and, meanwhile, obtains a good compromise between exploration and exploitation.

INDEX TERMS Reinforcement Learning, ε-greedy, exploration-exploitation, Q-learning, state aggregation.

I. INTRODUCTION

Reinforcement learning (RL) [1], [2] has been extensively used in many areas, such as machine learning, control engineering, non-linear systems, robotics [3]-[6], etc., for its powerful learning ability and good performance of online adaptation to complex nonlinear systems. RL methods deal with the problem that how an active agent can learn to approximate an optimal behavioral strategy by interacting with the environment. However, there exist several dilemmas for RL methods in developing practical applications. One major problem to solve is how to design the exploration strategy, which consequently contributes to better learning efficiency by balancing the trade-off problem between exploitation (take the most advantage of the experienced knowledge) and exploration (try to find a better policy with previously unexplored strategies).

In recent years, to tackle the dilemmas, many methods have been proposed. The exploration strategies are consist of two types of strategies: directed exploration and undirected exploration. Directed exploration is based on observing the learning process, while undirected methods often modify the probability distribution from random actions. Recently, ε-greedy strategy [7] and randomized strategy [8] have become the most popular undirected exploration strategies. And the balance between exploration and exploitation is usually tackled simply by a probabilistic exploration scheme, such as the ε-greedy methods, where each time the agent merely explores with a preoperational probability of ε to select between uniformly available actions in the action set and exploit based on the current policy in others (1-ε) at each step. A negative influence of the ε-greedy exploration scheme is that the learning time is exponentially proportional to the amount of states [9]. To increase the probability of exploitation over time, in the ε-decreasing method, the exploration rate initiates with a comparatively high value, and the exploration rate is decreased as the number of steps increases, which means the value of ε decreases as the experiment progresses, resulting in highly exploratory behavior at the start and highly exploitative behavior in the end. However, the decreasing rate of ε is difficult to set manually. Another well-known exploration scheme is the soft-max method [8], [10], where an action is selected with probability based on normalized exponential function. The temperature function is scheduled to decrease over time. When the temperature is high the actions tend to be selected...
uniformly. When it approaches zero, the selection probability for the action with a maximal value function is almost fully affirmative. These methods are popular due to their simplicity as well as the assurance of the convergence to the optimal policy. However, some research has shown the drawback of such exploration methods, where, especially in large complex problems, the parameters of these methods are difficult to be set manually.

Recently, some valid and near-optimal methods have been proposed, such as E3 [11], [12], R-MAX [13], [14], and so forth. In these algorithms, the exploration methods are embedded in the learning algorithm itself to solve the trade-off problem efficiently and implicitly. Whereas, some algorithms [11] explicitly handle the tradeoff problem by identifying the notion of experienced states, where these states have been visited a large amount of times. When in a familiar experienced state, the method evaluates the exploration and exploitation policy respectively. If the output of exploitation strategy is a $\epsilon$-optimal return, this policy is used; otherwise, the exploration policy is used for a long period. In contrast to handling the trade-off problem explicitly in alternative applied policies, some model-based algorithms [15] use a model for experienced states and apply a separate estimated empirical model for “unknown” states, or simply use a sole generative model for both kinds of states [16]. The pitfalls of these algorithms are that agents are encouraged to visit the unexplored state. This can also lead to abrupt changes in successive behaviors, and cause exponential dependence on the horizon time. In addition, such algorithms explicitly divide the learning process into the exploration and exploitation phases such that they may turn abruptly to a certain behavior.

To learn the optimal policy in a continuous environment, the other problem to solve besides experience collection is the data aggregation, especially when a group of similar experiences is identified. To develop novel algorithms which can learn optimal policies in continuous spaces, plenty of research efforts have been devoted recently. The research efforts usually tackle with discretizing continuous variables and transferring them into a new discrete version [17]. A rough discretization may lead to a coarse policy. On the other hand, too granulated discretization may lose the ability of generalization and increase the required training data size. Instead of taking direct approaches to state discretization, neural networks are used as function approximators to evaluate the state values in some researches [18]. Although the state discretization problems have been dealt with, it has the defect of convergence, and it is serious when dealing with complex state value functions. A linear combination is used to estimate the state value functions in tile coding [19]. Although it deals with the continuous state and real value action and has advantages of using less memory than partitioning the state space directly, choosing the size of the tiling and tiles is resemble to choosing partition size. Self-organizing map methods can partition the state space adaptively[20], [21]. However, these methods always require some time for computation to evaluate the vigilance parameter of all nodes. Hence, the aim of this paper is to address the dilemma of exploration strategy and adaptive state aggregation.

In this paper, the dilemma of state aggregation and exploration strategy is solved adaptively. To achieve the application of RL in real continuous environments, the adaptive state space segmentation algorithm based on a decision tree is proposed to avoid a rough discretization or redundant data. Additionally, this paper uses the adaptive $\epsilon$-greedy strategy to determine the probability to explore, and the value of $\epsilon$ relies on the computation of information entropy, instead of tuning the rate of exploration manually. To improve the efficiency and the quality of exploration, a Tabu list is used in this method. The proposed method obtains a good compromise between exploration and exploitation for reinforcement learning and partitions the state space into exclusive subspaces adaptively through simulation and some related discussions. The main contributions of the paper can be included as:

1) A systematic state aggregation method is introduced so that the problems in continuous state domain can be transferred efficiently into the discrete state space in valued-based reinforcement learning.

2) This paper provides an efficient experience collection scheme which can direct the agent to visit more informative states for learning acceleration.

The remainder of this paper is organized as follows: Section II introduces the method of adaptive state aggregation in detail, which includes discretizing a continuous state space by a decision tree structure. Section III represents a method to determine the value of $\epsilon$ by adjusting the information entropy, and a Tabu search for exploitation is illustrated. In Section IV, the details of simulation results show the availability and superiority of the EbIE-T method. Finally, Section V draws conclusions.

II. ADAPTIVE STATE AGGREGATION

Although RL techniques are useful for dynamic control tasks, the dynamics of a general system evolve in continuous state space due to continuous control efforts. This characteristic means that there are several practical problems in applying RL to a physical system because most RL algorithms are based on the assumption that the task is represented by discrete sets of states and actions. This problem is resolved by many RL methods by replacing discrete lookup tables that are indexed by states and actions with function approximators, which tackles continuous variables with several dimensions and can generalize across similar states according to an expert’s experience. However, it has been proved that merely replacing the tables may lead to learning failure, even for benign cases.
A. DECISION TREE for STATE AGGREGATION

To discretize a continuous state space, similar states are categorized into the same state. Meanwhile, continuous actions are also dispersed into separate or divergent discrete actions. Expertise in knowledge domains can be helpful to these processes, but a successful expert's experience is always hard to obtain or interpret. The purpose of the proposed method for state aggregation is to adaptively discretizes the state space based on a decision tree, without the need for expert experience. Initially, there is only one root node that corresponds to the whole state space. Sometimes, a small decision tree burgeons in advance from the root node. After some training processes, the root node or leaves can be divided into two child nodes, which are equivalent to two separate subspaces. After repeated iterations of the training process, the tree grows and the state space is split into exquisite subspaces.

During a training process, the agent obtains a sensory input vector \( u_t \), the most fitting leaf node is found and denoted as \( S_t \), covering \( u_t \). Then the agent selects an action according to action values which are recorded by leaf nodes. When the action is taken, the agent obtains the next sensory input vector \( S_{t+1} \) and a reinforcement signal or reward \( r_{t+1} \). The agent repeats the same action until it enters another leaf node. The sojourning period, during which the agent repeatedly takes the action in the same node, is called an epoch and is represented as \( \tau \). Every action value for all sensory input vectors in the epoch at time \( t+k \) is expressed as a semi-MDP:

\[
Q(u_{t+k}, a) = r_{t+k} + \gamma r_{t+k+1} + \ldots + \gamma^{t+k-1} r_{t+1} + \gamma^{t+k-1} \overline{V}(s'),
\]

where \( \overline{V}(s') \), defined as \( \overline{V}(s) = \max \overline{Q}(s, a) \), is the estimated optimal state value of the leaf node \( s' \). The estimated action value of the leaf \( S_t \) is updated as:

\[
\overline{Q}(s, a) = \overline{Q}(s, a) + \alpha [Q(s', a) + Q(s_{t+1}, a) \\
+ \ldots + Q(s_{t+k-1}, a)]/\tau - \overline{Q}(s, a)
\]

A state error, \( (u_t, e) \), is defined as the pair of sensory input vectors and the estimation error \( e = Q(u_{t+k}, a) - \overline{Q}(s, a) \), which is derived from (1) and (2). All estimation errors of these sensory input vectors are estimated and a list of the leaves is employed to record them before updating the estimated state-action value of \( S_t \). After updating the estimated state-action value, the leaf node decides whether to segment according to the distribution of sensory inputs and the magnitudes of estimation errors of the leaf node. If the length of the records is less than a pre-defined threshold, \( I_{th} \), it implies the estimated state-action values may not converge to a certain degree. If not, the mean average error value \( \mu_e \) and variance \( \sigma_e \) of the estimation errors recorded in the leaf node are employed to decide if it should be split or not. Then the estimation errors are used to update the estimated state-action values, that is, the estimated values are in proportion to the error value. Hence, the average value of the estimation errors are ought to approaches zero as time goes by. If the magnitude of the mean \( |\mu_e| \) is large to some degree, it implies the agent is still in learning progress and the leaf node should not be segmented. When \( |\mu_e| \) is less than the degree, the variance \( \sigma_e \) is employed to evaluate the accuracy of the current estimation. When \( |\mu_e| \) is close to zero, a small variance means most of the errors are small too, and the action values have been estimated accurately and the leaf node should not be pruned; on the other hand, a large variance implies errors are so diverged that a split may be essential for the improvement of estimation. In summary, a leaf node are ought to be split under the following three situations:

(a) The number of state errors exceeds a predefined threshold \( I_{th} \); for instance, 500 in this study;
(b) \( |\mu_e| \) is less than the threshold of mean average value \( I_{th} \);
(c) the variance \( \sigma_e \) of the estimation errors is larger than a predefined threshold \( \sigma_{th} \).

When a leaf is going to be split, another problem to deal with is that which dimensional axis is spanning the space and where it should be split. In this paper, we use the weighted T statistic to calculate the split dimension [22]. The state-errors in a leaf can be separated into two groups; one, the value is positive, which is denoted as \( g_1 \), and the other, the value is negative, which is denoted as \( g_2 \). In each group, the elements of the sensory input vector which are in the same dimension are considered as a set of statistical samples. This is denoted as \( X_{ij} \sim (\mu_{ij}, \sigma_{ij}) \), where \( \mu_{ij} \) and \( \sigma_{ij} \) represent the average value and the variance of dimension \( i \) of group \( g_j \), respectively. If \( X_{1j} \sim (\mu_{1j}, \sigma_{1j}) \) and \( X_{2j} \sim (\mu_{2j}, \sigma_{2j}) \) have similar \( \mu_{ij} \) and \( \sigma_{ij} \), it means the effect of state variables in dimension \( i \) is less on the estimation error. Then we use T statistic to evaluate the similarity between \( X_{1j} \sim (\mu_{1j}, \sigma_{1j}) \) and \( X_{2j} \sim (\mu_{2j}, \sigma_{2j}) \).

The \( t \) value of the T statistic is calculated by (3).

\[
t = |\mu_{ij} - \mu_{ij}|/\sqrt{\sigma_{1j}/n_1 + \sigma_{2j}/n_2}
\]

where the sample count of group \( g_j \) is written as \( n_j \). If the value of \( t \) is higher, it means there is less similarity between the two sample groups. The proposed method then sets accommodating with positive and negative errors. The dimension with the highest \( t \) value is chosen to be segmented. However, because the calculation of these state-errors uses the trajectory along which the agent moves in the state space, they may not be spread uniformly around the region that is defined by the leaf and can fall into a small region. This leads to undesired over-splitting, whereby the same dimension is always chosen to be split and leaves contain tiny fragments along that dimension.
To address this problem, a weighted T statistic is proposed. The t value of the weighted T statistic is calculated as:
\[
t_i = \text{range}_i \left[ \mu_{ij} - \mu_i \right] \sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}
\]  
(4)
where \text{range}_i is the range of dimension \( i \) of the leaf node.

Finally, the problem of where to split remains. Since \( \mu_{ij} \) and \( \sigma_{ij} \) are attainable, the density functions of the two groups are \( \frac{1}{\sqrt{2\pi\sigma_{ij}}} \exp\left(-\frac{(x - \mu_{ij})^2}{2\sigma_{ij}^2}\right) \). The roots of the equation can be found in [23],
\[
p(1)\sqrt{2\pi\sigma_{ij}} \exp\left(-\frac{(x - \mu_{ij})^2}{2\sigma_{ij}^2}\right) = p(2)\sqrt{2\pi\sigma_{ij}} \exp\left(-\frac{(x - \mu_{ij})^2}{2\sigma_{ij}^2}\right)
\]  
(5)
where \( p_i \) is the number of state errors for group \( j \) divided by the total number. The splitting value equals to the root between the means of the two groups. If the leaf is going to be split, the leaf converts into an interior node and the state action value and policy of the two new leaves will be inherited from their parent node.

B. PRUNING
As mentioned previously, the estimation errors for Q-learning decide where to split. The estimation errors always depend on the splits in the decision tree unless the estimation errors are too small to split. If there exists an inappropriate split, the estimation errors for other partitions may be influenced. As a result, when the estimation errors haven’t yet converged, especially during the earlier process, redundant leaves occurs. During exploration, some unimportant areas are scarcely visited when they are segmented and succeed in their parents’ dominating policy. Therefore, scales of confidence and vigor are defined to determine if these siblings should be pruned.

When the size of activation times on a node is large, there is more confidence in the action whose q-value is max, it is the optimal action at the node. Therefore, the confidence is in proportion to the number of activation times, and confidence of a node is written as:
\[
\text{confidence} = \frac{2}{1 + \exp(-c_{\text{con}} \cdot \text{active})} - 1
\]  
(6)
where a confidence constant is denoted as \( c_{\text{con}} \) and active is the activated times of the node. On the other hand, the number of episodes during which the node is inactive is defined as inactivity. At the start, inactivity of the activated nodes is set to zero. When an episode is finished, inactivity of all nodes increase by 1, and vigor of a node is denoted as:
\[
\text{vigor} = \frac{2}{1 + \exp(c_{\text{vig}} \cdot \text{inactivity})}
\]  
(7)
where a vigor constant is defined as \( c_{\text{vig}} \). When an episode is finished, all nodes of the decision tree are visited once. And inactivity of a leaf node will increase by 1. The confidence and vigor of the siblings are calculated when siblings of a leaf are both leaves. If vigor is less than confidence and the two siblings have the same the unique optimal actions, the siblings are split. Then the weighted sums of the child nodes are consist of q values, the action bias and the action disturbance are the weighted sum of the parent nodes according to their confidence.

III. ADAPTIVE EXPLORATION

The fixed ε-greedy method actually chooses random actions with a small fixed probability of \( \varepsilon \) and chooses the actions with the maximum Q-values with a probability of \( 1 - \varepsilon \), while the ε-decreasing method chooses the random action with a decreased probability over time [24]. The parameter \( \varepsilon \) has to set manually, and it won’t change over time. A strategic search which uses information entropy and a Tabu list is proposed to achieve a compromise between exploration and exploitation. The method adapts the ε-greedy method using a probability of state-action-dependent exploration and \( \varepsilon \) is determined using the information entropy that is calculated from the number of all actions taken in a state. Although the proposed method can explore a new trajectory using temporary information entropy, it inevitably falling into local optimum. If there is an extremely large space or a long distance from the starting point to the destination, the agent repeatedly translates its state in a local area. Therefore, a Tabu list of varying length of a Tabu list is used.

A. EXPLORATION BY INFORMATION ENTROPY
An adaptive ε-greedy strategy uses information entropy (EbIE), where the value of \( \varepsilon \) varies according to the learning progress instead of manual tuning. Radical fluctuations in the value estimates result in a high value for \( \varepsilon \) (exploration) while low fluctuations lead to a low value for \( \varepsilon \) (exploitation) [25]. The proposed method uses a state-action-dependent exploration probability to control the agent’s action-selection policy. The expected behavior is that the agent is more exploratory if a prevailing action appears at the start of the learning process, hence, the defined entropy is introduced in a state during the learning process. The amount of exploration is ought to be decreased until the knowledge of the agent becomes sustainable, which is indicated by very small or no temporal differences, \( \Delta Q(s,a) \). This adaptive behavior is achieved by computing the state-action-dependent exploration probability after each learning step. The computation of \( \varepsilon \) is as follows: In action space \( A = \{a_1, a_2, \ldots, a_k\} \), the probability of action \( a \) occurring in state \( s \) is denoted as:
\[
P(s,a) = \frac{N(s,a)}{N(A_s)}
\]  
(8)
where \( N(s,a) \) is the number of times action \( a \) occurs in state \( s \), \( N(A_s) \) is the number of all actions that occur in state \( s \) and \( P(s,a) = 0 \) if \( N(A_s) = 0 \). The entropy \( H(s) \) at state \( s \) is defined as:
\[
H(s) = - \sum_{a_s \in A(s)} P(s,a_s) \log P(s,a_s)
\]  
(9)
where \( 0 \leq i \leq N \) and \( 0 \log 0 = 0 \). It is obvious that \( H(s) \leq \log(N) \) when equality holds if and only if \( P(s,a) = 1/N \) for all \( i \). Then the normalization of the entropy \( H(s) \) is defined as:
\[
\overline{H(s)} = H(s) / \log(N)
\]
which is called metric entropy.

Taking entropy in the environment into account, the value of \( \varepsilon \) can be used as the quantity of \( \overline{H(s)} \) i.e.,
\[
\varepsilon = \begin{cases} 
(1 - \overline{H(s)})(1 - \exp(-|\Delta Q(s,a)|/N_\varepsilon)), & \text{if } |\Delta Q(s,a)| < C_H \\
1 - \overline{H(s)} & \text{otherwise}
\end{cases}
\]
where \( C_H \) is a small constant, \(|\Delta Q(s,a)| = \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))\) is derived from Q-learning and \( N_\varepsilon \) is the incremental episode counter which can be viewed as a temperature that causes the sufficient condition to be satisfied as the episode trials increase.

\( \overline{H(s)} \) is a measurement of the complexity of the action space. If \( \overline{H(s)} \) value becomes lower, the agent concentrates on some actions, but the agent must increase the probability of performing uniform selections. That is, when the value of \( \overline{H(s)} \) is low, the agent must perform more exploration. Therefore, defining \( \varepsilon = 1 - \overline{H(s)} \) increases the probability of performing exploration tasks. To prevent the agent from becoming trapped in long-term exploration, the factor \( 1 - \exp(-|\Delta Q(s,a)|/N_\varepsilon) \) in (11) guarantees convergence for long-term learning, as \( |\Delta Q(s,a)| \) decreases or \( N_\varepsilon \) increases. An exploitation task must be the last action in the learning process because the agent must be greedy in terms of the limit of infinite exploration (GLIE). The advantages of the adaptive exploration probability formula defined in (11) are that the agent does not need to carry out manual tuning and that it offers good performance. The simulation results for the proposed method are shown in the next section and the proposed algorithm is detailed below.

**Algorithm 1** The adaptive \( \varepsilon \)-greedy strategy

1. **Definition:**
2. \( N \) := the number of actions
3. \( C_H := a \) small constant threshold
4. \( N_\varepsilon := a \) number of incremental episodes
5. \( P(s,a) := a \) probability of action \( a \) occurring in state \( s \)
6. \( \overline{H(s)} := a \) entropy at state \( s \)
7. \( \overline{H(s)} := a \) metric entropy normalized by \( H(s) \)
8. **Initialization:**
9. Initialize \( Q(s,a) \) arbitrarily, e.g. \( Q(s,a) = 0 \) for all \( s,a \)
10. \( N_\varepsilon := 0 \)
11. \( \varepsilon := 1 \)
12. **For each episode**
13. \( N_\varepsilon := N_\varepsilon + 1 \)
14. Initialize start state \( s \).

15. **Repeat**
16. \( \xi := a \) random \( 0..1 \)
17. **If** \( \xi < \varepsilon \) **then**
18. Choose an action \( a \) uniformly.
19. **Else**
20. \( a := a \arg \max \beta \in N_a Q(s,b) \)
21. **End if**
22. Take action \( a \), observe reward \( r \) and successor state \( s' \).
23. \( \Delta Q(s,a) := \beta(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \)
24. \( Q(s,a) := Q(s,a) + \Delta Q(s,a) \)
25. Calculate the probability \( P(s,a) \) using formula (8).
26. Calculate the entropy \( H(s) \) and the metric entropy using formula (9), (10), respectively.
27. **If** \(|\Delta Q(s,a)|/N_\varepsilon \leq C_H \) **then**
28. \( \varepsilon = 1 - \overline{H(s)} \)
29. **Else**
30. \( \varepsilon := 1 - \overline{H(s)} \)
31. **End if**
32. \( s := s' \)
33. **Until** \( s \) is the terminal state
34. **End for**

**B. EbIE WITH A SHRINKING TABU LIST**

The concept of a Tabu search involves avoiding useless exploitation around a local area by forbidding or punishing moves that have been visited in the solution space [26]-[28]. The proposed method uses a Tabu list to record node-action pairs. After the epoch is finished and entering the next node, the node-action pair \((s_t, a)\) is recorded into the Tabu list if the node-action pair has not been recorded, where the node is denoted as \( s_t \) and the selected action is denoted as \( a \). The Tabu list is maintained on a first-in-first-out basis. The modified \( \varepsilon \)-greedy policy and the Tabu search are combined to produce a search strategy. When the generated random number is less than the threshold \( \varepsilon \), a random action is selected by agent \( a \). Otherwise, it applies a greedy policy. If \((s_t, a)\) are on the Tabu list, the values in the Q-table are temporarily set to negative infinity. That is, the precedence of the actions on the Tabu list is always the lowest. On the other hand, the performance of the state aggregation method is affected by the length of the Tabu list. If the Tabu list is short, a Tabu search gives no advantage. A long Tabu list inspires the agent to engage in more exploitation. Therefore, the length of the Tabu list shrinks with the total times of achieving the goal successfully. The proposed EbIE with a Tabu list method is summarized briefly as a flow chart in Fig.1.

A combination of exploration and exploitation methods is used to extend EbIE to the so-called EbIE-T method. In this method, the state-action-dependent exploration rate is...
updated adaptively according to EbIE, but selects random actions on the basis of the entropy in a state-action pair. A mean independence of the distribution of q values in state $s$ is achieved, which allows a more intuitive selection.

The range of the position is $[-1, 1]$ and the range of the velocity is $[-4, 4]$. There exist two actions to select in each state, and their values are +4 or -4 respectively. If the car is vertex on the left side, the velocity of it is zero, and the position is dented as -1. The immediate reward in all of the positions is zero, except when the car arrives at the vertex on the right. When the car reaches the destination, the immediate reward is in proportion to its velocity. The reward function is written as:

$$r = \begin{cases} 
\exp(-x) & x \geq 1 \\
0 & \text{otherwise}
\end{cases}$$  

(14)

The parameters for the simulation are listed in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>0.2</td>
<td>Threshold length</td>
<td>500</td>
</tr>
<tr>
<td>Discount rate</td>
<td>0.99</td>
<td>Threshold mean</td>
<td>0.5</td>
</tr>
<tr>
<td>Vigor constant</td>
<td>0.002</td>
<td>Threshold variance</td>
<td>0.001</td>
</tr>
<tr>
<td>Confidence constant</td>
<td>0.002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The learning rate or step size determines the degree of new information replaces old information, while the discount factor evaluates the importance of future rewards, these two parameters are used in conventional reinforcement learning. The confidence constant $c_{\text{con}}$ is used to calculate the confidence of the node while the vigor constant $c_{\text{vig}}$ is used to calculate the vigor of a node, the two parameters must be equal to decide whether the siblings are pruned. And the threshold length $l_{\text{th}}$, threshold mean $\mu_{\text{th}}$, threshold variance $\sigma_{\text{th}}$ are the thresholds of the number of records, mean average error and estimation errors recorded in the leaf node respectively, they are employed to decide whether to segment.

The performance of the weighted T statistics and the pruning strategies are compared, and the black line is the path of the state transition for the car. Fig.2 shows the results for the algorithms with weighted T statistics but no pruning (a), with T statistics and pruning (b) and with weighted T statistics and pruning (c). The vertical dimension is the velocity of the car, from -4m/s to 4m/s, and the horizontal dimension is the position of the car, from -1m to 1m. It is seen that the state partitions are fine along the path of the state transition. Other areas are coarsely partitioned, especially in the negative velocity region. Since the task is to climb the mountain and stop at the top, the velocity is close to 0. However, if the velocity is 0 before the car reaches the top, the car falls back into the valley. Therefore, the sensory inputs are sensitive near position 1 and the adaptive state aggregation partitions the state space more finely in this area.

The number of lean nodes for the three algorithms is compared in Fig. 3. The black line is the state transition path of the car climbing the hill. The car tries to climb the hill directly in the early period so the state space where the position and velocity are close to 0 is partitioned more finely. However, if the car explores an indirect path to climb the hill,
the state space where the position and velocity are close to 0 is seldom activated. The algorithm without pruning retains these leaves, so it has more leaves than the algorithms that use pruning.

![State partitions in the mountain car simulation](image)

**FIGURE 2.** The state partitions in the mountain car simulation

The performance of Tabu searches with a fixed length and shrinking length are also compared in Fig. 4. The x-axis is the initial length. In Fig. 4 (a), the y-axis represents the steps for the car to travel from the valley to the hilltop. In Fig. 4 (b), the y-axis is the velocity of the car as it climbs to the hilltop. Since the environment is more sensitive, the Tabu search with a fixed length significantly affects the car’s ability to climb the mountain, especially when there is a large initial length. This results in a bad performance, in terms of the number of steps and velocity. However, a large initial length is better for exploring the optimal path to climb the hill. Using a shrinking length algorithm, the velocity of the car is closer to 0 when the initial length is large.

The comparison between the adaptive state aggregation method and box method are listed in Table 2. All simulations use a TSSL with an initial length of 100. Since the action values in the state space have different complexity in different areas, the box method, which uniformly partitions the state space, is subject to sparse partition or dense partition. If the positional dimension is partitioned into 12 intervals and the velocity dimension is partitioned into 16 intervals for the state space for Q-learning, i.e., Q-Learning with a 12 x 16 box, there are 192 states in total and the velocity of the car is 0.53. Although the number of states is similar to that for the adaptive state aggregation, the performance, in terms of velocity, is poor. However, in order to have similar performance for velocity, Q-Learning with a 30 x 50 box must partition the state space into 1500 states, which is significantly more than is required for adaptive state aggregation.

![Comparison of fixed Tabu size and shrinking Tabu size in the mountain car simulation](image)

**FIGURE 4.** The comparison of fixed Tabu size and shrinking Tabu size in the mountain car simulation

**B. EXPLORATION WITH EbIE**

The simulation involves a mobile robot that searches for the goal in a maze, as shown in Fig. 5.

![Comparison of fixed Tabu size and shrinking Tabu size in the mountain car simulation](image)

**FIGURE 5.** The comparison of fixed Tabu size and shrinking Tabu size in the mountain car simulation
The range of the maze is 300 x 300, which is surrounded by walls and is partitioned into 100 x 100 state spaces in a grid-world. The agent starts from the blue spot in the upper-left corner. The aim for the agent is to reach the red rectangle in the center of the maze. The action space consists of four directions of movement (up, down, left and right) and in each step, the distance the agent can move is 5.0. All actions are imposed by Gaussian noises. The mean is set to 4 and the variance is set to 0.5 for this Gaussian noise. If the agent hits the walls or barriers as a result, it stands still and receives a punishment of -100. If the agent can reach the goal, a reward of +100 is given. Otherwise, the reward of each step is -0.01. The episode won’t be terminal unless the agent reaches the goal or the number of steps reaches 5000. The agent runs 1000 episodes per round. The learning rate is set to 0.85 and the discount rate is set to 0.9 for the Q-learning simulation.

The average number of steps that are required for the agent to arrive at the goal for each episode in 100 rounds of training are shown in Fig. 6. These curves represent the soft-max method using fixed temperature $T = 0.1$, action selections using $\varepsilon$-greedy with a fixed value for $\varepsilon = 0.3$ and the proposed adaptive $\varepsilon$-greedy method with a constant threshold $C_0 = 0.025$. The proposed method is considerably better than the others in terms of learning speed and convergence. The proposed adaptive $\varepsilon$-greedy method decreases the average number of steps to 1000 steps in the 343rd episode. The strategy using $\varepsilon$-greedy action selection with fixed $\varepsilon$ reaches the goal in the 502nd episode and the soft-max method does not reach the goal until the 670th episode. The proposed method also converges at around 92 steps, compared to 132 steps for the soft-max method and 173 steps for the fixed $\varepsilon$-greedy action selection method. As seen in Fig. 6, the proposed method converges with less noise (due to taking actions with Gaussian noise) than the other two methods. A comparison of the results in Fig. 6 shows that the proposed method performs best.

A complex maze environment is designed for a comparison between the proposed method and existing exploration strategies. An agent searches for the goal in a maze with hells, as shown in Fig. 7.

The maze with size 30 grids x 30 grids exists more than one hundred hells, which is not easy for agent to reach the goal. The agent has four actions for moving in one of the directions of the compass. The spot in the upper-left corner of the maze is the starting point for the agent. The goal is the yellow rectangle in the center of the map. If the agent takes an action and falls into any hell as a result, it starts from the upper-left corner again and receives a punishment of -1. When the agent reaches the goal, it receives a reward of +1. Otherwise, the reward of each step is -0.01. The episode is terminated when the agent reaches the goal or falls into the hells. The two parameters for the conventional Q-learning simulation, the learning rate is set to 0.8, and the discount rate is set to 0.98.

The average number of steps that are required for the agent to arrive at the goal are shown in Fig. 8. The four curves represent the soft-max method using fixed temperature $T = 0.1$; the $\varepsilon$-decreasing method with initial $\varepsilon = 0.3$, which decreases 0.00005 per round, leads a time-decayed exploration rate; action selections using $\varepsilon$-greedy with a fixed value for $\varepsilon = 0.3$; and the proposed EbIE method with a constant threshold $C_0 = 0.025$. Because the temperature parameter $T$ in the soft-max method requires knowledge of the likely action values, which is difficult to set manually, the result of this method is worst. The parameter setting for the fixed $\varepsilon$-greedy method can be easier, but the exploration rate is fixed over time, which leads to a mediocre result. As the improvement of the fixed $\varepsilon$-greedy method, the $\varepsilon$-decreasing method performs better, but the liner decreasing exploration rate may lead to slow convergence speed. In our method, the value of $\varepsilon$ varies according to the learning progress instead of manual tuning, which leads to the best result compared with the other three methods. A comparison of the results in Fig. 8 shows that the proposed method performs best, where the learning speed and convergence speed is better than the other three methods.
the pruning are compared. Fig. 10 shows the state partitions for the adaptive state aggregation. Fig. 10 (a) shows a state aggregation with weighted T statistics, but no pruning. Fig. 10 (b) shows a state aggregation with T statistics and pruning. Fig. 10 (c) shows an algorithm with weighted T statistics and pruning. All of the state partitions are coarse around the four corners and are fine along the walls. Since the state aggregation in (b) uses T statistics, there are many threadlike state partitions. Fig. 11 shows the curves for the number of leaves within the three algorithms. A state aggregation with weighted T statistics reduces the size of the decision tree more effectively than pruning. Pruning reduces the redundant nodes, regardless of whether the state aggregation uses weighted T statistics or T statistics.

Fig. 12 shows a comparison of a fixed length and a decreasing length. The x-axis is the initial length. In Fig. 12 (a), the y-axis represents the steps from the starting point to the goal. If the initial length is large, a Tabu search using the fixed length forces the agent to exploit, so more steps are required to approach the goal. Since the length of the TSSL shrinks as the number of successes increases, the agent finally exploits the learning and performs better. If the initial length is small, TSSL cannot explore an effective path and the agent requires more steps to reach the goal. Fig. 12 (b) shows the number of failures in approaching the goal. If the agent cannot approach the goal after the 3000th episode, the round is terminated and is denoted as a failure round. The y-axis in Fig. 12 (b) represents the number of failures before 20 successful rounds. If the initial length is small, an agent that uses the TSSL hardly explores the goal. Therefore, a large initial length is required if the goal is to be explored. However, excessive length results in over-exploration and greater learning time.

![Image of maze](image)

**FIGURE 9. The maze for the approaching goal simulation**

**TABLE III**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Threshold length</td>
<td>500</td>
</tr>
<tr>
<td>Discount rate</td>
<td>0.98</td>
</tr>
<tr>
<td>Threshold mean</td>
<td>0.5</td>
</tr>
<tr>
<td>Vigor constant</td>
<td>0.014</td>
</tr>
<tr>
<td>Threshold variance</td>
<td>0.03</td>
</tr>
<tr>
<td>Confidence constant</td>
<td>0.014</td>
</tr>
</tbody>
</table>

Firstly, the performance of the weighted T statistics and
requires a similar number of steps, it requires more states than adaptive state aggregation with weighted T statistics and pruning. However, although Q-Learning with a 17 x 17 box has fewer states, it requires more steps to approach the goal. If the state space is partitioned into 256 (16 x 16) states, the agent cannot explore the goal.

In order to verify the effect of pruning, state aggregation with and without pruning respectively is compared. The simulations were performed in two stages; the first being from the 1st episode to the 4900th episode and the second from the 4901st episode to the 5000th episode. The first stage is the same as the mountain car simulation and the second stage is also the same as the first stage, except for the starting point. In the second stage, the agent starts at a random corner of the maze. In Fig. 13, the y-axis represents the average number of steps required when starting from a corner. Both agents approach the goal without excessive exploration after the second stage and both methods perform similarly. State aggregation with pruning allows retention of experience and maximum pruning of the decision tree.

Table 4 shows a comparison of the adaptive state aggregation and the box method in the approaching goal simulation. All simulations use the TSSL with an initial length of 100. Although adaptive state aggregation using weighted T statistics and pruning significantly reduces the number of leaves, there is less difference in the number of steps required. Comparisons are made with Q-Learning using different resolutions, where the state space is segmented into 289 (17 x 17 box) and 400 states (20 x 20 box) on average. Although Q-Learning with a 20 x 20 box
The two curves in the middle are the sliding averages of the decision tree and the average state space cut respectively. The two methods both use an adaptive search strategy with a shrinking Tabu list. The results illustrate that the decision tree method is superior to the average cut in terms of the convergence of the learning speed and the number of steps. After 895 rounds, the average number of steps for the decision tree decreases to about 500 but the average state space cut requires about 4000 steps. In about 1500 rounds, the average number of steps for the decision tree method converges to about 245 while the average cut step method is still about 4000. The effect of different exploratory strategies on learning speed and step convergence is studied.

The average number of nodes that is generated is then compared. Fig.16 shows the average number of nodes that are generated by the decision tree in each round of 50 average times. The two curves represent the adaptive search strategy and the fixed $\varepsilon$-greedy strategy respectively. The number of nodes that is generated by the adaptive search strategy is less than which is generated by $\varepsilon$-greedy. In 3249 rounds, the number of nodes that is generated by the adaptive search strategy is about 600 while the number of nodes that is generated by the fixed $\varepsilon$-greedy is about 620. There is a difference because the Adaptive $\varepsilon$-greedy strategy learns faster than the $\varepsilon$-greedy strategy.

V. CONCLUSIONS
Since computation time complexity and memory size increase exponentially with the number of states, most efficient RL algorithms are merely applied in environments where the state is discrete or limited. This study uses algorithms to address the problem of online computation and continuous state space in RL, with a feasibly efficient exploration of the state space. The EbIE-T method uses an adaptive state aggregation approach that has a tree structure to transfer the continuous states into the discrete states efficiently. Meanwhile, to obtains a good compromise between exploration and exploitation, the adaptive exploration method with a Tabu list is introduced, which adjusts the value of $\varepsilon$ according to the learning process of the agent. Further, the experimental results prove the effectiveness and superiority of this method. In the future, we will determine how to explore the entire environment by more than one agent in parallel.
REFERENCES


