Learning to Select Relevant Perspective in a Dynamic Environment

Zhihui Luo, David Bell, Barry McCollum, and Qingxiang Wu

Abstract— **When an agent observes its environment, there are two important characteristics of the perceived information. One is the relevance of information and the other is redundancy. The irrelevant and redundant features which commonly exists within an environment, commonly leads to agent state explosion and associated high computational cost within the learning process. This paper presents an efficient method concerning both the relevance of information and the correlation in order to improve the learning of reinforcement learning agent. We introduce a new concurrent online learning method to calculate the *match count C(s)* and *relevance degree I(s)* to quantify the redundancy and correlation of features with respect to a desired learning task. Our analysis shows that the correlation relationship of the features can be extracted and projected to concurrent biased learning threads. By comparing the commonalities of these learning threads, we can evaluate the relevance degree of a feature that contributes to a particular learning task. We explain the method using random walk examples and then demonstrate the method on the chase object domain. Our validation results show that, using the concurrent learning method, we can efficiently detect redundancy and irrelevant features from the environment on sequential tasks, and significantly improve the efficiency of learning. After relevant features are extracted, the agent can remarkably accelerate its succeeding learning speed.**

I. Introduction

Reinforcement learning (RL) provides a fundamental framework for intelligent agents to improve their behavior through interacting with the environment [1]. In a delayed feedback and non-deterministic environment, RL proves to be an effective way to train an agent to perform at a high standard [2]. However, in real environments, the agent suffers from the consequence of redundant and irrelevant information which causes the state dimensions to explode. It is desirable for agents to develop an ability to generalize over the state space [3; 4]. One important step toward a higher level of state generalization is to find the most relevant aspects of a learning task [5]. Selecting observation perspective is a basic ability of the humans. When performing a task, even though we may see everything, we intuitively concentrate on the most relevant aspects for the current task. To develop such ability in an intelligent agent, it needs an algorithm to search for and evaluate the relevance degree of environment features. Furthermore, the agent also needs methods to utilize the discovered relevant features to assist task learning and performance.

Manuscript received Nov 29, 2007.

Zhihui Luo, David Bell and Barry McCollum are with the School of Computer Science, Queen's University Belfast, Belfast City, UK (e-mail: zluo02@qub.ac.uk, da.bell@qub.ac.uk, b.mccollum@qub.ac.uk).

Qingxiang Wu is with School of Physics and Opto-Electronic Technology, Fujian Normal University, Fuzhou, China (e-mail: qxwu@fjnu.edu.cn).

Environmental relevant feature evaluation and extraction is aimed at discover and remove as much as possible of the irrelevant and redundant information, and at identifying the relevant information from a desired task. Relevant feature extraction can be beneficial to learning by reducing the dimensions of task, compacting the size of the state space, and hence allowing learning to converge faster. Another potential benefit is that the discovered relevant features can be used as core learning elements for further learning on similar or related tasks. For example, the relevant feature can be used to generate options [6; 7], or as reusable knowledge [8; 9; 10].

We aim to develop a reinforcement agent that can automatically learn to select the most relevant state features in dynamic environments. We define a state feature as relevant if its information makes a contribution to the current learning; otherwise the state feature is irrelevant. And we define state features as redundant if they provide duplicate information for the learning. Both irrelevant and redundant features can be discarded intentionally by the agent to improve its learning.

In this paper, we will present a new online concurrent learning method that can distinguish core state features from the irrelevant and redundant features in the observed environment. Our method automatically evaluates the state features' relevance degree when the agent is performing a reinforcement learning task. After discovering relevant features, our algorithm removes irrelevant and redundant features from the ranked features list based on comparing commonalities between features in each state. Using the most relevant state feature, the agent forms a compact state representation of the current tasks and improves the learning efficiency for current and future related tasks in a dynamic environment.

II. Feature Correlation Definition

A. State Feature

In a dynamic environment which exhibits more than one feature, an agent perceives a lot of information. Not all of the acquired information is useful to achieve the task. For example, see figure 1, a robot can detect four different objects moving in the environment. At specific time *t*, the robot can see the status of these objects, which forms the state features s_t^1, s_t^2, s_t^3, and s_t^4. If only one of these objects is related to the rewarding task, how can the robot decide which of the objects is relevant? Further more, for some complex tasks in a dynamic environment; the real relevant features are usually dynamically dependent on the current progress of the task. The agent need to specify which feature is relevant to the current stage of learning. So it is desirable for the agent to find and select those state features that contribute to current task. In this paper, we combine the relevant feature discovery process with online reinforcement learning, so that our method can not only be beneficial for the current learning task, but also for later related learning tasks.

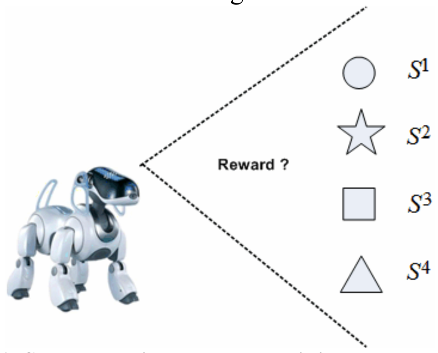

Figure 1. Sample environment containing several features

We consider an agent solving a problem with state space S and n state features. Suppose at the jth state $s_j \in S$, the learning task contains the features f_j^1, f_j^2, f_j^3 … f_j^n. These features represent different aspects within the environment, so the condition of each of these features can be used as sub-state of current state s_j, represented as s_j^1 s_j^2 s_j^3 … s_j^n. These state features determine the state dimension of the problem. So the jth state in state space S consisting n features is represented as:

$$s_j = \{s_j^1, s_j^2, s_j^3, \cdots s_j^n\} = \{s_j^i \mid i = 1,...,n\} \tag{1}$$

Note that in the following sections of this paper, we will omit state index j. We are not concerned with the state index j in this paper. So in the following sections, we will use a state with superscript $s^1, s^2, s^3, \cdots s^n$ to represent the corresponding sub-states for different features in state s.

B. Correlation Definition

Let x, y be the only two independent state features at state s, then state s can be defined as $s = (s^x, s^y)$ according to (1) . The learned policy for task S using both of the features is defined as:

$$\pi^*(s) = \pi^*(s^x, s^y) = \arg\max_{a \in A} Q(\mathrm{s}^x, \mathrm{s}^y, \mathrm{a}) \tag{2}$$

Similarly, we define the learned action policy which only referring to feature s^x as $\pi^*(s^x)$, and $\pi^*(s^y)$ referring to s^y in the same learning task. We can then define the correlation relationship of features based on comparing the learned policies.

Definition 1: relevance and irrelevance definition

In a learning task S with independent features s^x and s^y, if

$$\Pr\left(\pi^*(s^x, s^y) = \pi^*(s^x)\right) > \rho \tag{3}$$

$$\Pr\left(\pi^*(s^x, s^y) \neq \pi^*(s^x)\right) > \kappa \tag{4}$$

are satisfied, we say feature x is a relevant while feature y is irrelevant at state s in task S. In this definition, threshold ρ and κ are used to determine whether the probability is high enough.

Definition 2: redundancy definition

In a learning task S with independent feature s^x and s^y, if

$$\Pr\left(\pi^*(s^x, s^y) = \pi^*(s^x) = \pi^*(s^y)\right) > \omega \tag{5}$$

is satisfied, then we say s^x and s^y are redundant at state s in task S at level ω. In this definition, threshold ω is used to determine whether the probability is high enough.

The relevance and irrelevance definition is based on comparison of learned policies. Definition 2 implies that, if two features are redundant to the same task, they must be both relevant to the learning. Relevance is a necessary condition for redundancy. When more than one features providing duplicate information, we say they are redundant.

A straightforward approach to evaluate the definitions' probability is to use task performing experience to estimate the defined conditions.

$$\Pr\left(\pi^*(s^x, s^y) = \pi^*(s^x) \middle| \exp\right) > \rho \tag{6}$$

$$\Pr\left(\pi^*(s^x, s^y) \neq \pi^*(s^y) \middle| \exp\right) > \kappa \tag{7}$$

In (6) and (7), exp is the observed experience in task learning. Statistical analysis of the experience data can be used to evaluate the real probability of the correlation relationship.

III. CONCURRENT BIASED LEARNING

We aim to develop a criterion to measure the correlation relationship among features defined in the previous section. We can see from the definitions in section II.B, irrelevance and redundancy features are defined based on the learned action policy. It is intuitive to develop a method to compare the similarities among the features' policies. However, learning and comparing each feature's policy one by one is extremely time-consuming and resource intensive. In order to compare the policies more efficiently, we introduce a new reinforcement learning method called *concurrent biased learning*. This is a multi-thread learning method, in which each learning thread refers to one feature of the environment. If an agent intentionally focuses on part of these environmental features to learn a policy of a task, we call this method a *biased learning*; otherwise, if an agent uses all features that it perceives to learn a task, we call this *unbiased learning*.

We use Q-learning [11] to learn a policy. Suppose the agent performed an action a_{t-1} at state s_{t-1}, and after state transition, it reaches state s_t. The Bellman update function [1; 12] for unbiased learning is:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha[r_t + \gamma \max Q(s_t, a_t) - Q(s_{t-1}, a_{t-1})] \tag{8}$$

Using (8) to update the action-state value Q, the agent will converge to an optimal policy π^*. At each state s, the learned policy can be extract from the learned action state value: $a = \arg\max Q(s,a)$. If the agent learns a policy $\pi(s^i)$ with respect to one of the environmental features s^i, according to (1), we call this biased learning with respect to feature s^i. The biased Q-value can be denoted as $Q(s^i, a)$. Then the Bellman update function of the biased Q-value is:

$$Q(s_{t-1}^i, a_{t-1}) \leftarrow Q(s_{t-1}^i, a_{t-1}) + \alpha[r_t + \gamma \max Q(s_t^i, a_t) - Q(s_{t-1}^i, a_{t-1})] \quad (9)$$

Comparing the update functions (8) and (9), we know that biased learning is updated on state feature s^i while unbiased learning is updated on all the features.

To compare the policies, we equip the agent with all the biased and unbiased learning threads. When executing a task, the agent performs these biased learning threads simultaneously. We call this *concurrent biased learning*. These concurrent learning threads with different state features are updated from the experience of the same task. But each thread only concentrates on the changing aspect from its own prospect of view and ignores others. Differences among these concurrent learning threads provide useful information about the environment. These differences will be analyzed in next section.

IV. CORRELATION ANALYSIS

In this section, we analyze the information generated by the concurrent biased learning and develop a method to evaluate the relevance degree of state features. The basic idea of this method is to compare the learned action policy among biased learning threads. If there is a high probability that two threads will select the same greedy action at a certain state, then we propose that these two features provide redundant information (see definition 2) at this state. Otherwise, if there is a high probability that two threads provide different greedy actions at a certain state, then we can say that at least one of the state features is irrelevant to current learning (see definition 1).

In Q-learning, the updating of action state value always selects the greedy action a^* on current value state s. The greedy action for the unbiased learning thread is defined as:

$$\textit{Unbiased Greedy Action}: \ a^* = \arg\max_{a \in A} Q(s,a) \quad (10)$$

For a biased learning thread, the updating selects the greedy action with respect to current state feature s^i. The greedy action for the ith biased learning thread is defined as:

$$\textit{Biased Greedy Action}: \ a^{i*} = \arg\max_{a \in A} Q(s^i, a^i) \quad (11)$$

The greedy action actually represents the greedy policy at that state. If the greedy action provided by state feature is the same with the greedy action of unbiased state at time t, we record this matching as evidence. We define *match trace* $c(s)$:

$$c_{Mt}^i(s) = \begin{cases} 1 & \text{if } a^{i*} = a^* \\ 0 & \text{if } a^{i*} \neq a^* \end{cases} \quad (12)$$

The match trace records a value of 1 when a biased learning thread provides the same greedy action as the unbiased learning thread at time t in state s. Otherwise, if they are not the same, $c(s)$ records a value 0. We sum up the count trace across a certain period learning time and get a *match count* $C_M^i(s)$ of each state. It is a measure of how the biased learning threads coordinate with the unbiased learning thread.

$$C_M^i(s) = \sum_t^{t'} c_{Mt}^i(s) \quad (13)$$

It shows how many times a biased learning thread makes the same greedy decision with the current learning thread. Similarly to the definition of match matrix, we can define and calculate the *unmatched count* $C_N^i(s)$ that does not coordinate with the unbiased learning:

$$C_N^i(s) = \sum_t^{t'} c_{Nt}^i(s) \quad (14)$$

Using the two matrixes define above, we can calculate the *relevance degree* of a biased learning thread:

$$I_M^i(s) = \frac{C_M^i(s)}{C_M^i(s) + C_N^i(s)} \quad (15)$$

And the *irrelevance degree* of a biased learning thread is calculated as follow:

$$I_N^i(s) = \frac{C_N^i(s)}{C_M^i(s) + C_N^i(s)} \quad (16)$$

which is the number of times state s is visited. From equation (15) and (16), we have:

$$I_M^i(s) + I_N^i(s) = \frac{C_M^i(s)}{C_M^i(s) + C_N^i(s)} + \frac{C_N^i(s)}{C_M^i(s) + C_N^i(s)} = 1 \quad (17)$$

The sum of $I_M^i(s)$ and $I_N^i(s)$ equals 1. If one of them is known, value of the other degree can be easily calculated. We use $I_M^i(s)$ to evaluate the probabilities for definition (6) and $I_N^i(s)$ to evaluate the probabilities of definition (7).

V. ALGORITHMS

To automate the evaluation of the relevance degree, we combine the online Q-learning with the implementation of concurrent biased learning presented in this paper. Algorithm 1 begins with initialization to the required variables (line 1-4). Then the agent performs concurrent learning on unbiased and biased threads (line 10, 11). During the learning, the match count $C^i(s)$ value is updated based on the accumulation of the current acquired greedy policy (line 12). After a certain period of learning, the most relevant feature is selected (line 13, 14). The selected features can be used for further learning or indeed directly applied to current learning.

Algorithm 1:
Concurrent RL to find the most relevant feature

```
1. Initialize all ith biased learning thread
2. Initialize all $C^i(s)$ and $I^i(s)$ to assign 0
3. Use unbiased learning thread TC as default
4. Use policy $\pi$ : ε-greedy
5. Repeat (for each episode):
6.    Initialize state $s$
7.    While $s$ not terminal goal do:
8.       Choose $a$ from $s$ using policy $\pi$
9.       Take action $a$, observe $r, s_{t+1}$
         //Update state value of the unbiased learning thread:
10.      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
         //Update state value of EACH biased learning thread:
11.      $Q(s^i_t, a_t) \leftarrow Q(s^i_t, a_t) + \alpha[r + \gamma \max Q(s^i_{t+1}, a^i_{t+1}) - Q(s^i_t, a_t)]$
         //Update match matrix for EACH feature:
12.      IF $a_{t+1} = a^i_{t+1} : C^i_M(s_{t+1}) \leftarrow C^i_M(s_{t+1}) + 1$
         ELSE IF $a_{t+1} \neq a^i_{t+1} : C^i_N(s_{t+1}) \leftarrow C^i_N(s_{t+1}) + 1$
         //Calculate relevant degree for EACH feature:
13.      Update relevant degree $I^i_M(s)$ based on function (15)
         //Find and select the most relevant perspect:
14.      IF relevant degree $I^i_M(s)$ is MAX among all features
         THEN select feature $s^i$ as the current learning perspective in state $s$.
15.      Update state $s \leftarrow s_{t+1}$;
16.   End when $s$ is a terminal state
17. End when learning has converged across all states
```

Algorithm 1 implements the correlation analysis we described in section IV. After some period of learning, the relevance degree $I^i_M(s)$ for each feature is calculated. The algorithm will automatically select the most relevant feature for the current state.

Algorithm 2 evaluates the detected features across the task based on function 16. If the irrelevance degree of a feature exceeds the predefined threshold, this feature will be removed in current learning threads. Note that in algorithm 2, the irrelevance degree $I^i_N(S)$ is calculated on all states in a task. So once the agent gains enough experience to decide that a feature is irrelevant, it will ignore this feature in the whole process of task performance.

Algorithm 2:
Irrelevant feature elimination

```
         //Update match matrix and unmatch matrix for EACH feature:
12.      IF $a_{t+1} = a^i_{t+1} : C^i_M(s_{t+1}) \leftarrow C^i_M(s_{t+1}) + 1$
         ELSE IF $a_{t+1} \neq a^i_{t+1} : C^i_N(s_{t+1}) \leftarrow C^i_N(s_{t+1}) + 1$
         //Calculate relevant degree & irrelevant degree for EACH feature:
13.      Update irrelevant degree $I^i_N(S)$ based on function (16)
         //Select the most relevant perspective:
14.      IF irrelated degree $I^i_N(S) > \kappa$, a predefined threshold
         THEN remove this feature thread.
15.      IF only one feature exsits,
         select this feature as current learning feature
15.      Update state $s \leftarrow s_{t+1}$;
16.   End when $s$ is a terminal state
17. End when learning is stable across all states
```

Algorithm 2 calculates and eliminates the irrelevant features (line 13, 14). After irrelevant feature elimination, the agent will find the relevant state feature and make use of the selected feature in the current learning. Action policies will be generated only from the relevant states after relevant feature discovery.

VI. EXPERIMENTAL METHODOLOGY

In this section, we first test our algorithm in the random walk domain. We analyze and explain the recorded results. Then we compare the performance of traditional reinforcement learning with our current method in a chasing moving objects example.

A. Random Walk Experiment

1) Examples of Relevance and Irrelevance

We illustrate our definitions of relevant and irrelevant in a random walk example (See figure 2). In this setting, the agent A can move one step left or right or stay still. There are also two moving lights in this world, one emitting blue light marked B and the other emitting red light marked R. Both lights walk randomly one step to the left or right or stay still in the original position. The agent is equipped with light sensors that can obtain the distance and direction of these two lights. A hidden reward is linked to the red light. Only when the agent reaches a predefined distance from the red light will it receive a positive reward +10 and achieve this task. The blue object has no linked reward.

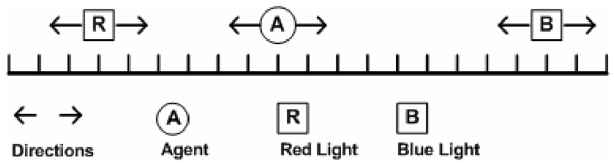

Figure 2: Example of relevant feature and irrelevant feature

From this description, we know the red light is relevant to the learning task while the blue object is irrelevant. But the agent has to discover the correlations of these objects in this dynamic environment. In order to observe the learning speed under different feature elimination parameters, we apply algorithm 2 in the experiment. The irrelevant feature is determined and eliminated based on evaluating all feature states.

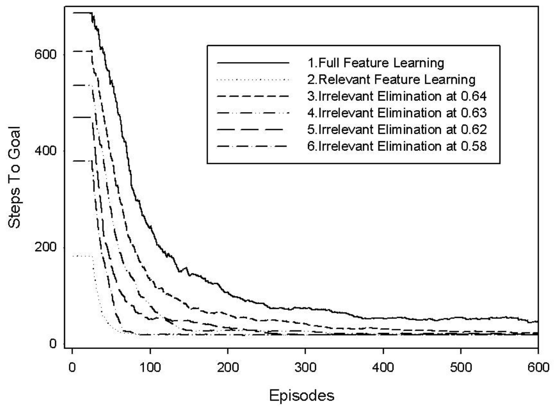

Figure 3: Learning curve of random walk example

Figure 3 shows that learning curve 2 converges much faster than learning curve 1. Curve 1 is an unbiased full feature learning, in which all features are taken into account regardless they are relevant or not, while curve 2 is obtained from the discovered relevant feature – the red object. The other curves show that with smaller irrelevant feature elimination threshold, the agent converges earlier and faster. Note that the agent may draw a wrong conclusion if we set the feature elimination threshold too small. In this example, if we set $\kappa < 0.5$ to eliminate irrelevant features, the agent will make a decision on what features are reverent at very early stage of learning. At that time, experience may not be enough to find the real irrelevant features. It is always safe to make such decisions after the current learning task converges. In this way, the current task learning cannot benefit from the feature discovery, but the discovered relevant features are still potentially be useful to succeeding related tasks.

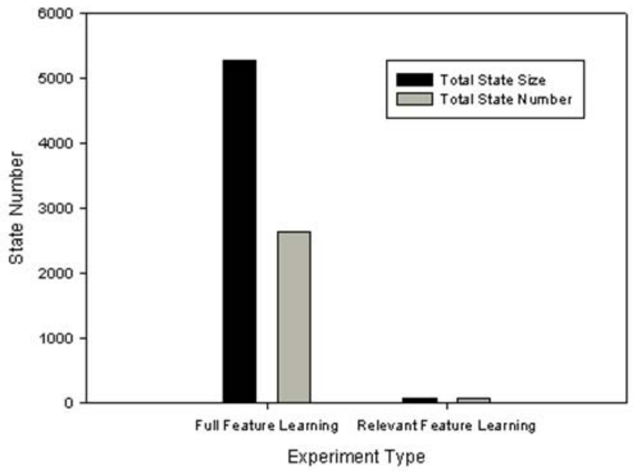

Figure 4: Recorded states of random walk example

Figure 4 shows that, if the agent cannot tell which feature is relevant, it will construct a much larger state space comparing to agent who learns only from the relevant feature. In this example (see figure 4 gray bars), learning with both features generates 2640 different states, while learning with the relevant feature only generates 88 states. Further more, for both features each state is represented by two light's coordinates, so its state space is double in size comparing to a state using only the relevant feature (see figure 4 black bars).

2) *Examples of Redundancy*

Similar to previous experiment, the agent can perceive 3 lights in this experiment: red light R , blue light B , and green light G . The agent can move left or right, but all light stay still in the random generated position. A positive reward +10 is located at a randomly generated position, but agent cannot directly detect the position of this reward.

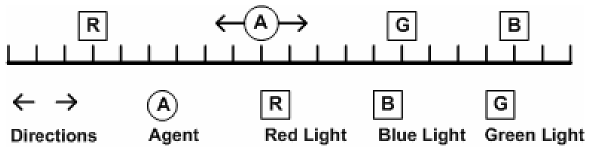

Figure 5: An example of redundant feature

In this experimental setting, either of the lights provides enough information to solve the problem. These lights features in the environment just provide duplicated information for states.

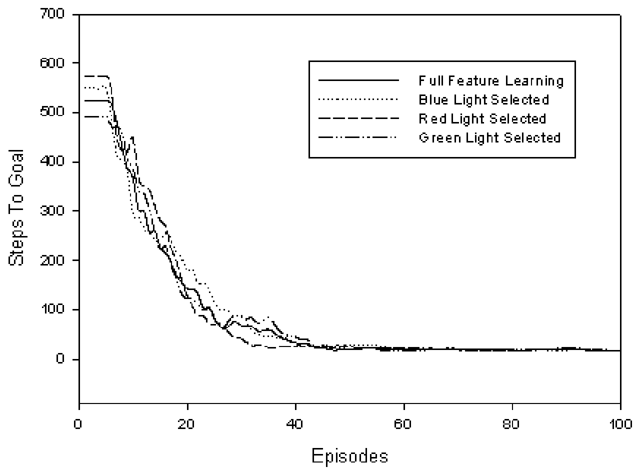

Figure 6: Learning curve with redundant feature

Figure 6 shows the learning curve of agents using algorithm 1. We can see that redundant features do not slow down or improve the current task learning, because every feature in the environment provides enough information to learn an optimal policy to get the reward. All 4 learning threads converge to stable at nearly the same speed. After sufficient learning at each state all of them generate exact the same learned policy.

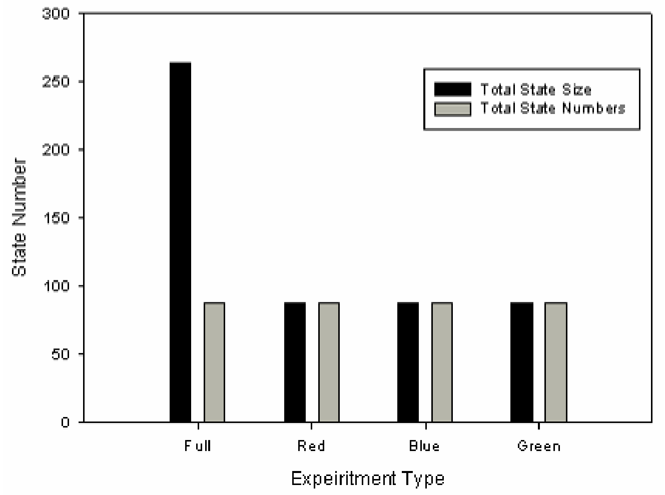

Figure 7: Recorded states of redundant feature

Figure 7 shows the recorded states in the random walk example with redundant features. In the state number graph (see figure 7 gray bars), we find that no matter what features the agent used, they generated exactly the same number of states, even using all learning features, the agent still generates 88 states which is the same with other features. But on the other hand, the state size (see figure 7 black bars) of full feature learning is 3 times larger than other states. The reason is that each full feature learning state contains the status of all three learning features. Hence removing redundant feature can reduce state size. It will also potentially reduce the computational requirement for comparing the state values at each learning step.

B. *Chasing Objects Experiment*

Chasing objects is a common scenario in the robotics area. In this section, we simulated this in a discrete grid world. This world is composed of 10*10 grids. There are 3 objects in this world marked object ①, object ②, and object ③. All objects can move one step randomly either in vertical, horizontal or diagonal directions. They can also choose to stay still in the original position. The agent in this world also can move one step randomly in each direction. It knows the exact locations of all objects.

The task for the agent is to follow and catch these objects one by one in a predefined sequence. We define the task for agent to catch these objects in a sequence of ①, ②, ③. Only when the agent performs this sequential task correctly, will it receive a positive reward +10 at object ③. In this example, the environment is highly dynamic due to the movement of three target objects and complex as there is only one delayed positive reward. So using simple Q-learning, the learning speed will be quite slow. We apply algorithm 1 to find the relevant feature, and then compare the learning speed before and after using our feature discovery method.

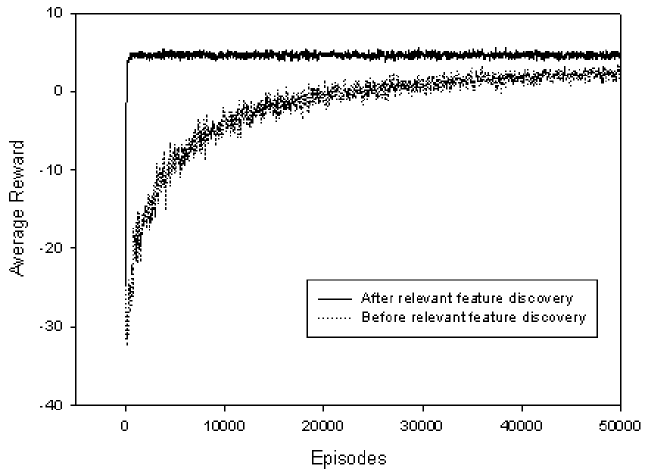

Figure 8: Learning before & after relevant feature discovery

Figure 8 compares the learning efficiency of learning the agent before relevant features discovery and after relevant feature discovery. The graph shows that after relevant features being discovered in each state, the agent can learn significantly faster than when using all features. An agent using relevant state features converges in the initial 500 episodes while an agent using full state features slowly converges after over 50000 episodes.

Table 1: Relevance degree at different stages of learning

Task Stage	**Chase ①**	**Chase ②**	**Chase ③**
Average value of $I_M^i(s)$			
$I_M^{objcet1}(s)$	★0.548	0.323	0.358
$I_M^{objcet2}(s)$	0.317	★0.672	0.325
$I_M^{objcet3}(s)$	0.291	0.307	★0.886

Table 1 summarizes the average relevance degree for all three features at different stages of learning. At each stage, the maximal relevant feature is marked with a star sign. It can be clearly seen that our method successfully distinguishes the most relevant object at each stage of learning. Object ① has the highest relevance degree at learning stage chasing object ①, and object ② at stage 2 so on. Overall, object ③ has the highest relevant degree at stage 3. The reason is that this part of the states is close to the reward and converges earlier than other states. We believe that, if the agent uses the knowledge discovered in table 1 to assist learning, it will perform even better (See figure 9).

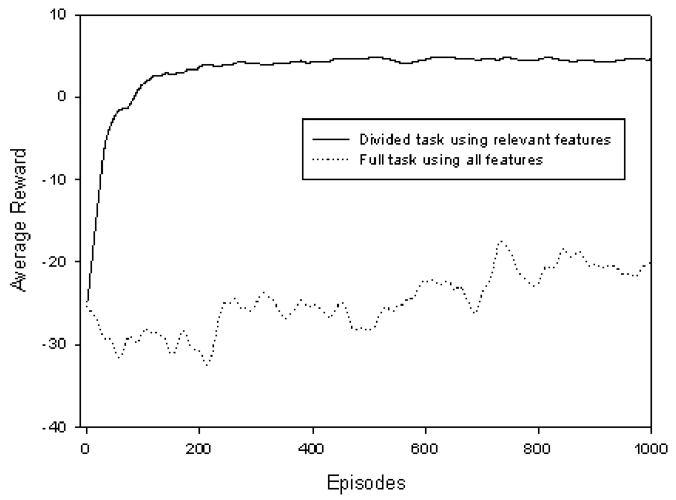

Figure 9: Compare knowledge discovery on task

If the agent uses the discovered knowledge in table 1 to perform the task, it can focus only on the most relevant features at different stages of learning. The learning results are compared in figure 9. This shows that, using an relevance observing method on this task, the learning is not only significantly faster but also smoother in nature than simple Q-learning.

VII. DISCUSSION

In the experiments outlined in the previous section, we show that our relevant feature discovery method can successfully discover the relevant features in a dynamic environment. The automated relevant feature discovery algorithm also can significantly increase the learning speed of the agents. Although our methods are tested on small grid world examples, we believe this is an import step toward higher levels of state generalization. Experimental results in section VI.B shows promising results.

From these experiments, we find that after the relevant features are discovered, the agents form a much smaller state space. We use the big O notation to describe the computation complexity of the agent's state space, and we use n to denote the number of learning features in the task and k to stand for the state size of a learning feature. Suppose in a dynamic environment, there is no constrain between these n set of features. Then the scalability of the agent's unbiased learning thread is $O(k^n)$. On the other hand, the state space for one of agent's biased learning thread is $O(k)$. The agent has n piece of biased learning threads, so the overall scale for all these learning threads is $O(n*k)$ which is a linear function of variables n and k . We know that $O(k^n)$ grows exponentially in respect of the features number n , while $O(n*k)$ is only a linear growth. For example, supposed in a fully dynamic environment, there are 5 independent features and each has 10 states. The traditional unbiased reinforcement learning forms a state space with 10^5 numbers of states. If the agent uses our biased learning threads to search for relevant features, it forms additional 5 learning threads which in sum have 5*10 states. We can see $10^5 >> 5*10$. From this analysis, we know that the computational complexity of relevant feature discovery is trivial comparing to traditional reinforcement learning. Our experimental results in the previous section conforms our analysis. These analyses also show the potential improvement of using only the most relevant feature.

VIII. RELATED WORK

Prior research exists on the topic of discovering relevant state features in reinforcement learning. One example of this reported by Jong and Stone [13]. They define irrelevance based on the assumption that there exists some action that achieves maximum reward regardless of the state feature Y at state s. They use a Bayesian approach to estimated the probability that Q(x,y,a)=V(x). The method can find an ignorable feature y. However, from their definition, it is possible that feature y is not irrelevant (which is relevant) for the learning, but just redundant for the learning. In their experiment on a Taxi world, either passenger source or destination position provides enough information to solve the task. These two variables are redundant to each other, although one of them can be ignored. In contrast, our method defines relevance based on comparison of learned action policies which discover both relevance and redundancy relationships among state features. The concurrent reinforcement learning algorithm evaluates the relevance degree of each efficiently.

Other related work is presented by Jonsson and Barto [14]. They apply the UTree [15] to form option context based state abstraction. This work makes the UTree algorithm suitable for state abstraction on option framework. However, this work required a predefined structure of options to be provided before the abstraction. We believe our method is one important way toward automated option generation.

An early work on this topic is named UTree developed by McCallum [15; 16] . The UTree algorithm records all experience instances based on action-perception-reward triple. This method connects predecessor and successor experience in the transition chain. A tree structure is used to organize these recorded instances. By statistically evaluating the value of the fringe of the UTree, useful features can be discovered. This method requires agent's to remember the chain of instances, and hence it potentially generates a very large experience memory. Especially in highly dynamic environments, both of the instances chain and the tree will require a large memory. Compared to UTree, our method handles the dynamic environment well by using concurrent learning. Also, the UTree method does not concern information correlation among features, whereas our method makes clear definitions on the correlation relationship among features.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we present a relevant feature discovery method for reinforcement learning. We define relevant, irrelevant and redundant relationships among features based on comparing action policies of the RL agent. Subsequently, we develop an online concurrent learning method to evaluate the correlation of state features. Our experimental results suggest that using most relevant features, the agent has remarkable improvements in relation in learning speed. Furthermore, this method also has good potential to be used to develop hierarchical abstractions upon the task. We believe the relevant feature extraction is an important step toward higher level of knowledge abstraction.

In the future study, we will estimate the performance of our algorithm in more complex and realistic tasks. We also find that concurrent learning method can possibly be applied to multi-agent learning, and this can also be a topic of our future work.

REFERENCES

[1] R. Sutton, and A. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[2] S. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall Series in Artificial Intelligence, 2003.

[3] A.G. Barto, and S. Mahadevan, Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems: Theory and Applications, 2003.

[4] P. Tadepalli, R. Givan, and K. Driessens, Relational Reinforcement Learning: An Overview, International Conference on Machine Learning, 2004.

[5] L.P. Kaelbling, T. Oates, N. Hernandez, and S. Finney, Learning in Worlds with Objects AAAI Stanford Spring Symposium on Learning Grounded Representations, 2001.

[6] R.S. Sutton, D. Precup, and S.P. Singh, Intra-Option Learning about Temporally Abstract Actions, Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc. , San Francisco, CA, USA, 1998, pp. 556 - 564

[7] T. Croonenborghs, K. Driessens, and M. Bruynooghe, Learning Relational Options for Inductive Transfer in Relational Reinforcement Learning, The 17th Annual International Conference on Inductive Logic Programming 2007.

[8] G.D. Konidaris, and A.G. Barto, Building Portable Options: Skill Transfer in Reinforcement Learning. Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007.

[9] M.E. Taylor, and P. Stone, Representation Transfer for Reinforcement Learning, In AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development, November, 2007.

[10] Z. Luo, D. Bell, and B. McCollum, Skill Combination in Reinforcement Learning, In proceeding of the 8th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'07), Birmingham, UK, 2007.

[11] C. Watkins, and P. Dayan, Q-Learning. Machine Learning, 8(3-4):279--292, 1992.

[12] R.E. Bellman, Dynamic Programming, Princeton University Press, 1957.

[13] N.K. Jong, and P. Stone, Towards Learning to Ignore Irrelevant State Variables, The AAAI-04 Workshop on Learning and Planning in Markov Processes, San Jose, CA, 2004.

[14] A. Jonsson, and A.G. Barto, Automated State Abstraction for Options using the U-Tree Algorithm, In Advances in Neural Information Processing Systems 14., 2001.

[15] A.K. McCallum, Reinforcement Learning with Selective Perception and Hidden State, Department of Computer Science, University of Rochester, Rochester, New York, 1995.

[16] A.K. McCallum, Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks, From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior, Cape Cod, Massachusetts, 1996.