Model Learning and Knowledge Sharing for a Multiagent System With Dyna-Q Learning

Kao-Shing Hwang, Senior Member, IEEE, Wei-Cheng Jiang, and Yu-Jen Chen

Abstract—In a multiagent system, if agents’ experiences could be accessible and assessed between peers for environmental modeling, they can alleviate the burden of exploration for unvisited states or unseen situations so as to accelerate the learning process. Since how to build up an effective and accurate model within a limited time is an important issue, especially for complex environments, this paper introduces a model-based reinforcement learning method based on a tree structure to achieve efficient modeling and less memory consumption. The proposed algorithm tailored a Dyna-Q architecture to multiagent systems by means of a tree structure for modeling. The tree-model built from real experiences is used to generate virtual experiences such that the elapsed time in learning could be reduced. As well, this model is suitable for knowledge sharing. This paper is inspired by the concept of knowledge sharing methods in multiagent systems where an agent could construct a global model from scattered local models held by individual agents. Consequently, it can increase modeling accuracy so as to provide valid simulated experiences for indirect learning at the early stage of learning. To simplify the sharing process, the proposed method applies resampling techniques to grafting partial branches of trees containing required and useful experiences disseminated from experienced peers, instead of merging the whole trees. The simulation results demonstrate that the proposed sharing method can achieve the objectives of sample efficiency and learning acceleration in multiagent cooperation applications.

Index Terms—Decision tree, Dyna-Q, model sharing, multiagent system.

I. INTRODUCTION

REINFORCEMENT learning (RL) takes actions in terms of exploration and exploitation in turn and then evaluates the rewards resulting from actions taken to search for an optimal policy. RL agents do not need to have prior knowledge about an environment initially [1]. The agents learn an optimal policy by a series of trial-and-error process of interacting with the environment so that a long learning time is needed. Therefore, sample efficiency is important in many RL applications for the agent can learn an effective policy in very few actions [2].

There are, in general, two approaches to achieving this goal. The first approach introduces the state aggregation method into model-free method [3]–[5], which learns the value function without using a world model. The state aggregation method generalizes the continuous state information that has the same value function. The second method develops a model-based algorithm that builds the transition probability function and reward function by fully use of a limited amount of experiences and can learn a better policy with fewer environmental interactions [6]. The model-free methods have been shown to learn the policy more slowly than model-based methods, though they are much simpler and are not affected biases in the design of the model. The Dyna architecture is a model-based method that is extended from RL architectures [7], [9], and it includes policy learning and an internal world model. However, how to build the internal world model is always ignored in the studies of Dyna architectures such that simple lookup table methods are used widely for environmental modeling. The tabular model resolves the problem about how to decide the grid resolution in a continuous space. A higher resolution requires more learning time but the model is more accurate, whereas a lower resolution needs less learning time but the model cannot approximate the environment accurately. When an RL agent acquires an accurate model in the application domain, it can use that model to perform value iterations by indirect learning, which performs the same learning process as direct learning but takes simulated experiences generated by the model, instead of real experiences.

Many researches have extended RL algorithms to multiagent systems, where agents can take decentralized structures to deal with tasks, to solve more complex problems [10], [11]. Agents usually evaluate their own sophistication levels of knowledge and share only rich and helpful experiences to partners. Therefore, other agents with less knowledge can take advantage of the ones with more experiences via sharing processes. The mechanism that peers deliberately share experiences with each other applied to a model-free RL is called policy sharing [12]–[14]. As well, the mechanism applied to model-based RL for environmental modeling is referred to as model sharing [15]–[17].

Whereas, there are some challenges in extending the model-based RL to the applications of multiagent systems. First, the properties of environments can be classified into deterministics and stochastics. When learning tasks are under a deterministic environment, the influence of the transition probability can be ignored. Instead, agents always transit onto next states with
some degree of uncertainty in a stochastic environment such that transition probability should be brought into model learning methods appropriately. Furthermore, agents need to explore the environment exhaustively and frequently in order to obtain an accurate virtual model approximating the environment. This process is quite time-consuming and costs tremendous computational power. To decrease learning effort and save time, the agents should share their model information between peers. That is easy to accomplish if agents have the same partition pattern in the state space or state aggregations so as to share their model or policy information straightforwardly. For example, the sharing is done by simply combing weightedly individual value functions and frequencies of visits to states with the selected peers’ throughout the whole state space. If each individual model held by robots is with a heterogeneous structure, how to share the corresponding information is a profound problem.

In this paper, models in the learning agents are constructed by discrepant decision trees, respectively. Although model sharing can be treated technically as tree merging, most of the proposed methods are to merge the whole trees; that is, the methods need to transplant the whole decision tree to other agents [18]. Whereas, not all nodes of a tree have helpful model information. In this paper, a model learning method based on tree structure is presented to achieve sample efficiency in task domains with continuous state spaces. The purpose of this model learning scheme is to automatically generate various resolutions by using a decision tree and to approximate the transition probability between two successive states. Based on the model learning method, several model sharing methods for multiagent systems are proposed. When learning agents have acquired sufficient experiences to precisely model a portion of the environment, they should share out their experiences to help others construct their models, even the latter scarcely or never explore that area of the state space. The proposed sharing methods assume the agents learn policies in a deterministic and stochastic environment occasionally and share information between heterogeneous models. These agents use tree structures to construct environmental models and share their experience under consideration of the information stored in leaf nodes so as to alleviate the loads on data transfer between agents as well as to save the computational time in sharing process.

This paper is organized as follows. Section II introduces the background, RL theory and Dyna architectures. The proposed model learning method based on tree structure is illustrated in Section III. Section IV introduces three successive sharing algorithms which extend the tree-model techniques to multiagent systems. In Section V, the simulations are conducted to show the validity of the proposed methods. The results demonstrate the applicability and effectiveness of the algorithms. Finally, the conclusions are drawn in the last section.

II. BACKGROUND

A. Reinforcement Learning

RL is a sub-area of machine learning, in which a learning agent takes actions in an environment so as to maximize the long-term reward [1]. The agents have no prior knowledge about the environment, and so they use the trial-and-error method to gather experience and perform the value iterations. In the interaction with an environment, the agent perceives current states \( s_t \in \mathbb{S} \), the state space, from environment and makes a decision action \( a_t \in \mathbb{A} \), the action set. After taking action \( a_t \), the state of the environment transfers from the current state \( s_t \) to next state \( s_{t+1} \) and the environment feeds back a reward \( r_{t+1} \), which evaluates whether the action is good or not. If the environment returns a good reward, then the agent will strengthen the decision action; otherwise, the agent will weaken the decision action. After the agent obtains many experiences to train the policy, the agent could use it to make a series of optimal action sequences to achieve the goal.

One of the most important breakthroughs in RL is Q-learning, which is an off-policy method that can be run on top of any strategy wandering in a finite Markov decision process (MDP). It uses the experiences to approximate the optimal value function, from which one can construct an optimal policy. Specifically, Q-learning can search an optimal action-selection policy for any given finite MDP. It learns an action-value function, Q-function, which ultimately gives the expected utility of taking a given action, \( a_t \), in a given state, \( s_t \), and following the optimal policy thereafter. When such Q-function is learned, an optimal policy can be constructed by simply selecting the action with the highest value in each state. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring any adaptations. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy [19]. The learning rule of the Q-function is listed as

\[
Q(s_t, a_t) = Q(s_t, a_t) + \beta (r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))
\]

where \( \beta (0 \leq \beta \leq 1) \) is the learning rate and \( \gamma (0 \leq \gamma \leq 1) \) is the discount factor. The learning has been proven to converge after the agents have had many interactions with the environment [19].

B. Dyna Architecture

A Dyna architecture is a model-based method which is extended from the RL architecture including policy learning and an internal world model [7]. The approximated model uses current states \( s_t \) and actions \( a_t \) as inputs, then the next states \( s_{t+1} \) and rewards \( r_{t+1} \) as outputs. The learning agent uses real experiences which have been gathered from the environment to build the virtual model and update its value function, a process called direct RL. After the virtual model has approximated the environment, it can simulate virtual experiences to obtain additional policy learning, called indirect RL or planning. A Dyna architecture coordinates a Q-learning algorithm to form a Dyna-Q framework including planning, acting, model learning, and direct RL. The direct RL is eventually a one-step Q-learning, where the model learning is implemented to record the experiences, \((s_t, a_t, s_{t+1}, r_{t+1})\). The agent selects
the state-action pairs randomly that have been experienced. The model uses the state-action pairs as inputs and predicts the next state and reward as a virtual experience.

### III. Environmental Modeling

The proposed method is based on a Dyna-Q architecture but uses a tree structure for environmental modeling. The tree-model is for generating virtual experiences to increase extra value iterations. Therefore, the proposed agent includes two interleaving processes, direct learning and indirect learning, i.e., planning. In direct learning, the agent perceives the environmental information, called continuous states $i_t$ and selects an action $a_t$. Then the agent transits to the next continuous states $i_{t+1}$ and receives a reward $r_{t+1}$. The continuous state-action pairs, $(i_t, r_t)$, are inserted into the set $I_R$. The Q-learning aggregates continuous states, $i_t$ and $i_{t+1}$, to discrete states, $s_t$ and $s_{t+1}$, and uses the experience, $(s_t, a_t, s_{t+1}, r_{t+1})$, to update the policy. The tree-model uses the experience, $(i_t, a_t, i_{t+1}, r_{t+1})$, to approximate the environment directly. In indirect learning, the tree-model substitutes for the environment and $N$ continuous state-action pairs, $(i_t, a_t)$, are retrieved from the set $I_R$. When the retrieved continuous state-action pairs are fed to the tree-model, the variation of next continuous states, $\Delta i_a$, and the rewards, $r_2$, are predicted. The next simulated continuous states is derived from $i_{t+1} = i_t + \Delta i_a$ and the virtual experience, $(i_t, a_t, i_{t+1}, r_2)$, is to speed up learning. Fig. 1 shows an example of a tree-model that splits the state space into four regions. The difference action in each region will produce the difference variation of continuous states, $\Delta i_a$ and the rewards, $r_2$.

#### A. Establishment of World Model

In the beginning, the tree-model only has a root node that corresponds to the entire state space. After several iterations in direct learning, the tree-model can be split into some child nodes that correspond to subspace. In direct learning, the agent perceives the continuous states $i_t$ from the environment and selects an action $a_t$. After taking the action, the agent observes the next continuous states $i_{t+1}$ and receives a reward $r_t$. The variation between two continuous states is defined as

$$\Delta i_a = i_{t+1} - i_t$$  \hspace{1cm} (2)

where $a$ denotes the selected action.

A variation vector is a two-tuple vector composed of variations of continuous state and the reward, $(\Delta i_a, r_t)$. All vectors are evaluated and recorded in the leaf nodes. The online cluster method is introduced to classify variation vectors. Due to the diversity of the vectors in each dimension, the normalization process must be taken prior for the classification process. When the first vector arrives, it forms the first cluster center, $M_1^a$; the number of vectors contained in the cluster, $N_1^a = 1$; and the number of the clusters, $C_a = 1$, recorded in the leaf node. When a next variation vector arrives, similarity between the variation vector and the existing cluster centers are measured by Euclidean distance to find the highest. If the highest similarity is larger than the threshold, $\delta_a$, a new cluster will be created; otherwise, the variation vector will activate the cluster with highest similarity. When a variation vector activates a cluster $j$, the cluster information will be updated. The information includes the number of vector, $N_j^a$ and the moving average of the variations of state transition and rewards. The update function is as follows:

$$N_j^a = N_j^a + 1$$

$$\Delta i_j^a = \left( \frac{\Delta i_j^a \times (N_j^a - 1) + \Delta i_a}{N_j^a} \right)$$

$$r_j^a = \left( \frac{r_j^a \times (N_j^a - 1) + r_t}{N_j^a} \right)$$ \hspace{1cm} (3)

where $\Delta i_j^a$ is the average of state variation vectors and $r_j^a$ denoted averaged rewards; $j$ denotes the activated cluster; $j = 1, 2, \ldots, C_a$.

After updating the information of the activated cluster, the leaf node needs to be tested whether to split or not. If the number of the variation vectors, $N_j^a$, is smaller than a threshold, $N_{th}$, which is a predefined constant, the leaf node does not need to split; otherwise, the next condition is introduced to evaluate whether to split or not. After classifying a variation vector, each cluster should be mapped onto the state space, as shown in Fig. 2. In each cluster, elements of continuous states in the same dimension are regarded as a set of statistical samples. This is written as $X_{\Delta i_j^a} \sim (\mu_{\Delta i_j^a}^a, (\sigma_{\Delta i_j^a}^a)^2)$, where $\mu_{\Delta i_j^a}^a$ and $(\sigma_{\Delta i_j^a}^a)^2$ are the means and the variances of elements of continuous states in dimension $i$, respectively. The proposed algorithm uses T-statistic [4] to select the splitting dimension. The T-statistic can be used to evaluate the similarity between two clusters in each dimension. If two clusters, $j_1$ and $j_2$, have similar means and variances in the dimension $i$, it means that the two clusters cannot be split easily. The $t$-value of $T$-statistic in dimension $i$ is calculated by

$$ t_i = \left| \frac{\mu_{\Delta i_{j_1}}^a - \mu_{\Delta i_{j_2}}^a}{\sqrt{\frac{(\sigma_{\Delta i_{j_1}}^a)^2}{N_{j_1}} + \frac{(\sigma_{\Delta i_{j_2}}^a)^2}{N_{j_2}}}} \right|$$ \hspace{1cm} (4)
where $\mu_{ij}^a$ and $\mu_{ij}^b$ are the sample means, $(\sigma_{ij}^a)^2$ and $(\sigma_{ij}^b)^2$ are the sample variances, and $N_{ij}^a$ and $N_{ij}^b$ are the number of variation vectors in clusters, $j_1$ and $j_2$.

If the means and variances of two clusters are close to each other in dimension $i$, then the $t$-value would approach 0. A higher $t$ value in the dimension $i$ implies less similarity of distribution between the two sample sets of clusters, $j_1$ and $j_2$. The dimension $i$ with the highest $t$-value is chosen and a split should be taken if $t$-value is over the threshold value $t_{th}$, and vice versa.

The T-statistic may have a blind spot and cause an unsuitable split. As depicted in Fig. 3, sampling points are marked by blue triangles, red circles and green squares. In dimension 1, the blue triangles and the green squares are two clusters that are used to decide the split point. In dimension 2, the red circles and the green squares are two clusters that are used to decide the split point. It is obvious that dimension 2 will have a higher $t$-value than dimension 1, and so this leaf node will be split at Split point 2. To avoid such situation, the algorithm should take the widths of the remaining ranges in a dimension after a split into consideration in evaluating which dimension should be split. The evaluated function is called the weighted T-statistic and is rewritten as

$$W_{ti} = \left(\text{range}_i, \left| \mu_{ij_1}^a - \mu_{ij_2}^b \right| \right) \left/ \sqrt{\frac{(\sigma_{ij_1}^a)^2}{N_{ij_1}^a} + \frac{(\sigma_{ij_2}^b)^2}{N_{ij_2}^b}} \right.$$

where $W_{ti}$ is a scalar value of the weighted T-statistic function; range$_i$ means the low and upper bounds of the range of dimension $i$ of the leaf node.

If the split point gets a larger $t$-value but the range is small, the $t$-value will be decreased. In contrast, if the split point gets a smaller $t$-value but the range is large, the $t$-value will be increased.

The next step is how to decide the split point in the dimension $i$. Since the means and variances of continuous states are calculated in the splitting dimension, which is decided by weighted T-statistic, the split point is evaluated by

$$\text{split point}_i = \mu_{ij_1}^a - \sqrt{\frac{(\sigma_{ij_1}^a)^2}{N_{ij_1}^a} + \frac{(\sigma_{ij_2}^b)^2}{N_{ij_2}^b}}$$

$$= \left(\mu_{ij_1}^a \times \sqrt{(\sigma_{ij_1}^a)^2} + \mu_{ij_2}^b \times \sqrt{(\sigma_{ij_2}^b)^2}\right) \left/ \sqrt{\frac{(\sigma_{ij_1}^a)^2}{N_{ij_1}^a} + \frac{(\sigma_{ij_2}^b)^2}{N_{ij_2}^b}} \right.\right.$$ (6)

After splitting the leaf node, the leaf node translates into an internal node and two new leaf nodes inherit the samples from their parent node.

### B. Tree-Model Recalling

In indirectly learning, the agent replaces the environment by the tree-model, which takes the continuous states and actions as input and searches leaf nodes according to the continuous states. After the corresponding leaf has been found, the clusters with input action which resides the leaf and contains maximal number of vectors contained is chosen. That is, the cluster which has highest activation frequency in clustering is used for prediction. The equation for activation frequency is defined as (7), which indicates that the probability is proportional to the number of activations in a cluster; that is, the cluster most frequently activated has the highest possibility of generating predicted variations

$$F(\overline{w}_{ij}^a | i, a) = \left\{ \begin{array}{ll}
 \max_j \left( N_{ij}^a \right), & j = 1, 2, \ldots, Ca \text{ if } N_{ij}^a \neq 0 \\
 \text{None} & \text{if } N_{ij}^a = 0 
\end{array} \right.$$

(7)

where $\overline{w}_{ij}^a$ is the cluster that is the predicted result. The mean of variation vectors, $\overline{w}_{ij}^a$, is the result in search by taking action $a$ in continuous state, $i$.

### C. Planning

The objective of the planning is to minimize as quickly as possible the temporal error defined in

$$\Delta Q(s_t, a_t) = |r + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)|.$$ (8)

In indirectly learning, the value iteration follows the tree-model to learn the policy to be improved. When an agent evaluates the state-action pairs under a policy according to a model, it is called planning or indirect learning. In the Dyna framework, the agent interacts with the environment and retains the experience at the same time. When planning is performed, the agent uniformly selects a stored state-action pair to refine the $Q$-value. The state-action pairs are previous experiences. The algorithm then updates the action-values by the randomly selected experienced state-action pairs.

In the proposed method, the agent retrieves the number of $N$ continuous states, $i$, from the set, $I_N$, and the action-values are updated by the tree-model. Before proceeding to value iterations, two kinds of action function methods, random action and action selection function (9), are used to select action, $\tilde{a}$, for comparison. A random action is uniformly selected from the action space, and the action selection function according to the current information is defined as follows:

$$\tilde{a} \leftarrow \arg\max_{a_t \in A} \left(1 - \lambda \right) \Delta Q(s_t, a_t) + \lambda \sqrt{\log(C_{\text{Node}(i)})}$$ (9)

where $\lambda \in [0, 0.5]$ is a constant; $C_{\text{Node}(i)}$ is the total number of planning times for a continuous state, $i$, on a certain leaf node; and $C_{\text{Node}(i, a_t)}$ is the number of planning times in the leaf.
In [2], the model considers the variation between the current probabilities between two states after an action has been taken. A. Sharing Under Compulsion (SuC)

In the SuC method, each agent voluntarily shares its knowledge about the environment to other agents. Since the agent uses a decision tree for environmental modeling, the model information collected from experiences in interaction with the environment is actually stored in leaf nodes. When an agent finishes an episode, it broadcasts its model information to resample simulated samples as sharing information. The model information from the agent’s queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model. The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. A cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.

Other agents receive and save the information in their own sharing queues until they terminate an episode. When an agent terminates an episode, it retrieves the sharing information from its queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model.

The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. If a cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.

Other agents receive and save the information in their own sharing queues until they terminate an episode. When an agent terminates an episode, it retrieves the sharing information from its queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model.

The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. If a cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.

Other agents receive and save the information in their own sharing queues until they terminate an episode. When an agent terminates an episode, it retrieves the sharing information from its queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model.

The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. A cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.

Other agents receive and save the information in their own sharing queues until they terminate an episode. When an agent terminates an episode, it retrieves the sharing information from its queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model.

The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. A cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.

Other agents receive and save the information in their own sharing queues until they terminate an episode. When an agent terminates an episode, it retrieves the sharing information from its queue, if it is not empty. The agent then uses the sharing information to resample simulated samples as sharing experiences to grow its own tree-model.

The first problem is how to decide the number of samples that should be generated according to each shared node. Because there may be many clusters in a shared node and each cluster has a different number of samples, the numbers of generated samples of a cluster, \( N_j \), are decided by (10). Therefore, to generate more samples from the larger clusters, which hold more samples, the element activity \( j \) is introduced. In (11), \( j \) is the number of clusters. The numerator of activity \( j \) means the number of samples collected in cluster \( j \) and the denominator is the summation of samples collected in all clusters. A cluster has more samples, the activity of the cluster is larger; otherwise, the activity of the cluster is smaller. \( N_u \), which is predefined, is the number of samples that should be resampled in a shared node. In (12), if the number of samples of a cluster is smaller than \( N_u \) and \( \mu = range_{i,U} - range_{i,L} \), where \( range_{i,L} \) and \( range_{i,U} \) are low and upper bounds of dimension \( i \) of a leaf node, respectively.
where \( X \) represents samples, \( \mu \) is the mean of samples, \( \sigma \) means the standard deviation of the samples, and \( k \) is a constant.

The intervals in each dimension include lower and upper bounds. In a cluster, the sums of continuous states and the square sums of continuous states in the same dimension, \( i \), are used to evaluate the intervals by counting the means and the variance. The intervals of each cluster are derived as follows:

\[
\mu_{a,i}^2 = X_{a,i}^2 / N_{a,i} \tag{14}
\]

\[
(\sigma_{a,i}^2)^2 = (X_{a,i}^2 / N_{a,i}) - (\mu_{a,i}^2)^2. \tag{15}
\]

According to (13), when \( k \) has a value of 2, approximately 75% of the data are in the range of \([\mu_{a,i}^2 - 2(\sigma_{a,i}^2), \mu_{a,i}^2 + 2(\sigma_{a,i}^2)]\). Nevertheless, uniform sampling in a certain range may generate some samples beyond the scope that a node represents, as shown in Fig. 5. In Fig. 5(a), the samples are generated within the interval, but samples cannot be generated in the interval in Fig. 5(b)–(d). Fig. 5(b) shows that the interval is over the range of the leaf node on the left side. In Fig. 5(c) the interval exceeds over on the upper bound of the range, and Fig. 5(d) is over on both bounds. To solve this problem, the ranges of the leaf node are introduced to limit the lower and upper bounds of intervals. According to Fig. 5, the intervals of the lower and upper bounds of each dimension \( i \) are as follows.

\[
\mu_{a,i}^2 - k(\sigma_{a,i}^2)^2 \leq \text{range}_{a,i} \]

\[
\mu_{a,i}^2 + k(\sigma_{a,i}^2)^2 \leq \text{range}_{a,i} \]

If \( k = 2 \) then

\[
\Rightarrow \mu_{a,i}^2 + k(\sigma_{a,i}^2)^2 \leq \text{range}_{a,i} - \mu_{a,i}^2
\]

If \( k = 2 \) then

\[
\Rightarrow \mu_{a,i}^2 + k(\sigma_{a,i}^2)^2 \leq \text{range}_{a,i} - \mu_{a,i}^2
\]

\[
\text{LowBound} = \mu_{a,i}^2 - 2(\sigma_{a,i}^2)^2
\]

\[
\text{UpperBound} = \mu_{a,i}^2 + 2(\sigma_{a,i}^2)^2
\]


When the intervals are decided, the samples are generated according to the intervals. The resampled samples in each dimension by the intervals are the simulated continuous states, \( i_t \); and actions, \( a \), are selected from the action space. Then the next continuous state is derived as follows:

\[
i_{t+1} = i_t + \Delta_i \tag{16}
\]

where \( \Delta_i \) means the variation of continuous state after taking the action, \( a \). The shared experiences, \((i_t, a, i_{t+1}, \rho_i^j)\), are produced, and \( \rho_i^j \) is the mean of rewards.

After creating shared experiences, these samples are passed through the decision tree, and fall into different leaf nodes. Fig. 6 shows an illustration of experience sharing in a leaf node on a demanding agent. In Fig. 6(a), the rectangle represents a leaf node of the agent with various samples, labeled by circles, triangles, and squares, prior to sharing. In Fig. 6(b), the agent receives the model information from peers, and the dotted rectangles denote shared nodes from experienced agents. In the dotted rectangles, asterisks are the shared experiences resampled by the shared nodes.

The asterisks should be classified into clusters, so the online cluster method is introduced. Due to the diversity of the samples in each dimension, the normalization process must be taken prior to the process of classification. When a sample arrives and the bold rectangle does not have any cluster, it forms the first cluster center. When the next sample arrives, it tests similarity between the samples and the existing cluster centers to find the highest cluster. Similarity between two samples is measured by the Euclidean distance.

The shared experiences, \((i_t, a, i_{t+1}, \rho_i^j)\), calculates the variation of continuous state by (2). The online cluster method is introduced to classification the variation vector to each cluster. After classifying the samples labeled with asterisks, the clusters originally in the leaf node become more informative and new clusters may be created, as shown by the diamonds in Fig. 7, which has four categories, triangles, circles, squares, and diamonds.

The category where there are less than \( N_i \)h samples is not considered when the leaf node is split. The means and variances are calculated by (14) and (15) to find a split point. The statistical result of each cluster is written as \( X_{i,j} \sim (\mu_{i,j}^2, (\sigma_{i,j}^2)^2) \), where \( \mu_{i,j}^2 \) and \( (\sigma_{i,j}^2)^2 \) are the means and variances of the continuous state of cluster \( j \) in dimension.
i, respectively. The proposed algorithm uses the T-statistic to select the split dimension and to evaluate similarity of the mean in each dimension between two clusters. The way of tree induction with shared experiences is identical to the one introduced in Section III. Fig. 8 illustrates the splitting result after creating shared experiences. The leaf node is split at Sp1 first; the main categories are “triangle” at right side and “circle” at left side. The leaf node is next split at Sp2, and the main categories are “circle” on the right side and “square” on the left side. The splitting point Sp3 is the same. If the t-value is smaller than a threshold, t_{th}, the process of splitting will be stopped.

B. Sharing Under Request for Unvisited State-Action (SRUS)

Similar to interactions in human society, when an agent perceives a continuous state it takes an action, referred to as continuous state-action pairs. If a pair occurring in an agent introduces a larger error for temporal difference learning, it then broadcasts that state-action pair to other agents. When an agent receives that pair and can categorize it onto a leaf node in its own tree, it checks whether or not there is sufficient knowledge about that area corresponding to the fitted leaf. If there isn’t, the agent tags that area as an unknown area and broadcasts a request to other agents for help. This communication process is shown in Fig. 9. The unknown area must satisfy at least one of the following conditions.

Condition 1: \[ N_{total} = \sum_{j=1}^{C_{a}} N_{j} < cN_{th} \]

Condition 2: Continuous state \[ i = (x_1, x_2, \ldots, x_n) \]

\[ \bigcup_{j=1}^{C_{a}} \left\{ x_i \mid x_i \notin \left[ \mu_{i,j}^a - k\sqrt{(\sigma_{i,j}^a)^2}, \mu_{i,j}^a + k\sqrt{(\sigma_{i,j}^a)^2} \right], i = 1, 2, \ldots, n \right\} \]

In condition 1, c is a constant and N_{th} is a threshold, which means that the sum of samples in the leaf node needs to be sufficient. Condition 2 is confidence ranges, where \( \mu_{i,j}^a \) and \( (\sigma_{i,j}^a)^2 \) are the means and the variances of continuous states in dimension i. These means and variances are calculated by the sums of continuous states and the square sums of continuous states of each cluster in leaf nodes. If one element of continuous states is not located in the confidence ranges of all clusters, then condition 2 is satisfied.

After the agent broadcasts its request composed by a continuous state-action pair to the other agents, the agents that receive the continuous state-action pair search their own model and return the information of leaf nodes of tree model whose covered area overlaps the one of the received pair. In Fig. 10, the black point is the continuous state-action pair; the green part is a shared node; the dotted rectangle represents the overlap between the found leaf node and the shared node. Before sharing the information, the shared nodes have some clusters, but the leaf node has no clusters or few.

If the number of samples of each cluster in the shared nodes is smaller than threshold N_{th}, the cluster will be ignored. After the small clusters are ignored, the remaining clusters are used to resample. Equations (10) to (12) are applied to decide the number of samples that should be generated in each remaining cluster. The samples are generated in the same way as by SuC. Fig. 11 depicts an example resampling samples according to one cluster in a shared node. In Fig. 11(a), the samples of one
cluster are created and only the samples staying in the portion of the rectangle with dotted edges should be considered. If the samples of a cluster are generated as in Fig. 11(b), where no resampled one falls within the leaf node, that cluster should be ruled out.

The samples generated by each cluster in a shared node and all the samples existing in that node should be classified together, but such process of classification is expensive. The resampled samples by the same cluster in a shared node still hold the same properties. In other words, the samples generated by the same cluster have the same variation vectors, \( v_{ai,j} \), so they can form the new cluster. The new cluster includes the information, the number of samples of each new cluster, the means of the variation of the samples, the mean of reward of the samples, the sums of samples, and the square sums of samples. The specifications of the new clusters are depicted in Table II. The proposed method is modified to merge the clusters that exist in the leaf nodes, produced by the shared nodes, as shown in Fig. 12.

![Fig. 12. Illustration of clusters merging between shared node and leaf node.](image)

### Table II: Specification of the New Clusters

<table>
<thead>
<tr>
<th>Model information</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{a}^i )</td>
<td>Number of samples of each new cluster</td>
</tr>
<tr>
<td>( \Delta r_{ij}^t )</td>
<td>Means of the variation of the samples</td>
</tr>
<tr>
<td>( \Delta \mu_{ij}^t )</td>
<td>Mean of reward of the samples</td>
</tr>
<tr>
<td>( \sum { x_{ij}^t } )</td>
<td>Sums of samples in each dimension</td>
</tr>
<tr>
<td>( (\sum { x_{ij}^t })^2 )</td>
<td>Square sums of samples in each dimension</td>
</tr>
</tbody>
</table>

Similarity between two clusters is calculated by the Euclidean distance, as follows:

\[
d = \min \left( \sum_{i=1}^{n} \left\| M_{i,j_1}^a - \Delta r_{ij_1}^t \right\|^2 + \left\| M_{n+1,j_1}^a - \Delta r_{j_2}^t \right\|^2 \right)^{\frac{1}{2}} \tag{17}
\]

where \( j_1 = 1, 2, \ldots, C_j^i \), \( j_2 = 1, 2, \ldots, C_j^m \) are the number of clusters in the leaf nodes and \( C_j^a \) is the number of clusters produced according to the shared nodes. \( M_{i,j_1}^a \) and \( M_{n+1,j_1}^a \) represent the center points of each cluster, which belong to the leaf node. In addition, \( \Delta r_{ij_1}^t \) and \( \Delta r_{j_2}^t \) are the means of variation of samples and the mean of reward of samples stored in each new cluster, and the mean of reward is considered the \( n+1 \)th dimension. Similarity should be examined in pairs between the clusters in the leaf node and new clusters produced by the shared nodes. If the highest similarity, \( d \), is larger than the threshold, \( \delta \), the cluster \( j_2 \) forms a new cluster and joins into the leaf node. Otherwise, the cluster \( j_2 \) is merged with cluster \( j_2 \) of the leaf node with the highest similarity, and the information should be updated as follows:

\[
N_{total} = N_{j_1}^a + N_{j_2}^a \tag{18}
\]
\[
\Delta \mu_{ij_1}^t = \left( \frac{N_{j_1}^a \times \Delta \mu_{j_1}^t + N_{j_2}^a \times \Delta \mu_{j_2}^t}{N_{total}} \right) \tag{19}
\]
\[
\Delta r_{ij_1}^t = \left( \frac{N_{j_1}^a \times \Delta r_{j_1}^t + N_{j_2}^a \times \Delta r_{j_2}^t}{N_{total}} \right) \tag{20}
\]

The sums of samples and the square sums of samples should also be updated. The next step is deciding whether or not to split the tree model. After the leaf node has been combined with the shared nodes, the largest cluster, which has the maximum number of samples, is selected as the activated cluster. The activated cluster tests the distribution of each dimension by (4). Each dimension can select a maximum \( t \)-value, so the numbers of \( n \) \( t \)-values are calculated. If the \( t \)-values of dimensions are smaller than the threshold \( t_{th} \), then the dimensions of tree model do not consider splitting; otherwise, the dimensions are retained. The remaining dimensions have the weighted \( t \)-values calculated by (5), the largest weighted \( t \)-value is selected and (6) is used to calculate the splitting value.

### C. Sharing Under Request for Ranges (SRR)

The SRUS method decreases the time in classification, but some inappropriately shared nodes may exist, as shown in Fig. 13. The colored rectangle denotes a shared node. If the dotted portion of the rectangles overlap partially the leaf node in the way shown in the figure, the samples generated by shared nodes may cause improper split points. The shared samples, dots in the figure, may crowd around rims of the rectangle, the rim on the left-hand side in the example, and possibly cause a split around shared samples. That split leads to a threadlike node.

![Fig. 13. Example of shared experiences resampled by a shared node may lead to an inappropriate split.](image)

The SRR method is proposed to avoid improper split points. This method is derived from the SRUS method. When an agent receives a continuous state-action pair; it searches its own model and the continuous state-action pair falls into the unknown range of the model. The agent with unknown ranges then broadcasts a request to other agents for help and the unknown ranges are evaluated the same as by the SRUS method. The agent sends the unknown ranges with the action to the other agents for help. An example is shown in Fig. 14(a), where the agents receive the unknown ranges, marked by the
V. SIMULATIONS

In this section, first a simulation, Mountain car, demonstrates the proposed model learning method, and Gaussian noise is attached to the working tasks to increase the complexity. The proposed method is compared with the original Q-learning [19] and Dyna-Q [7], which is implemented by tabular models, to verify its performance. Second, two scenarios, a searching goal problem and a mobile robot in a maze, demonstrate the model-sharing methods for simulation. The robots explore the environment and learn an optimal or near optimal policy. When the robots have some knowledge of the environment, they share their knowledge to each other. The sharing methods lead the robots to go to the goal as quickly as possible without bumping into any obstacle in the space. The simulation results verify the efficiency of the proposed methods.

A. Mountain Car

In this simulation, there is a car in a valley, which does not have enough power to climb the hill directly, as shown in Fig. 15. It must first move up the left side to acquire potential energy, and then to the right side, using the potential energy to climb the hill. The dynamic equation is

\[ x_{t+1} = x_t + \dot{x}_t + 1 = \dot{x}_t + 0.001 a_t - 0.0025 \cos(3x_t) \]  

where \( x_t \) is the position of the car and \( \dot{x}_t \) is its velocity.

The state space of the problem has two dimensions, its position and its velocity. The position boundary is \([-1.2, 0.5]\) and the velocity boundary is \([-0.07, 0.07]\). In each state, three actions are allowed: throttle forward (+1), throttle reverse (-1), or zero throttle (0). All three actions have attached Gaussian noise. The noise uses three actions as mean and the variance is set at 0.1. When the car has reached the top of the right side, it has achieved its objective. The immediate reward is -1 everywhere, except when it reaches the right boundary, the immediate reward is 1. The parameters are listed in Table III.

The value of \( \delta_a \) is measured from interactions with environment. In the case of the mountain car, 2000 samples after an action has been taken are collected to calculate the variance between two continuous states, and the different action is calculated separately in the same way. The means, \( \mu_a \), and standard deviation, \( \sigma_a \), of the variations are computed along each dimension, where \( i \) is the number of dimensionality, \( i = 1, 2, \ldots, n \). In addition, the scalar rewards are calculated in the same way to obtained the means, \( \mu_a \), and standard deviations, \( \sigma_a \),

\[ \delta_a = (\sigma_1^a + \sigma_2^a + \ldots + \sigma_n^a + \sigma^a) / H, H \in Z^+ \]  

where \( H \) is the accuracy control parameter and \( Z^+ \) is a positive integer. The more complicated the working environment is, the larger the value of \( H \) that should be selected and the more accurate the tree-model is.

As depicted in Fig. 16, the average numbers of steps for the agent to arrive at the goal of each episode is 40 rounds of training. There are four curves represents the results from Q-learning, original Dyna-Q, and the proposed Dyna-Q with two kinds of planning functions, respectively. The proposed agent can reduce the average steps to 600 obtain its objective in the 23rd and 9th episodes by selecting actions at random and by using the action selection function, respectively. The Q-learning takes until the 67th episodes and the agent with

![Fig. 14. Example of shared range. (a) Shared nodes overlap with the received range. (b) Shared nodes sent back to the agents.](image)

![Fig. 15. Mountain car task.](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>0.7</td>
<td>Discount rate</td>
<td>0.8</td>
</tr>
<tr>
<td>Exploring rate</td>
<td>10%</td>
<td>Planning Number(N)</td>
<td>5</td>
</tr>
<tr>
<td>( \delta ) (throttle forward)</td>
<td>0.03078</td>
<td>( \delta ) (zero throttle)</td>
<td>0.02923</td>
</tr>
<tr>
<td>( \delta ) (throttle reverse)</td>
<td>0.03241</td>
<td>( N_h )</td>
<td>5</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>0.15</td>
<td>( H )</td>
<td>2</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Fig. 16. Simulation results in searching goal problem. x-axis is the number of episodes and y-axis is the number of step taken to reach the goal.

**TABLE IV**

**COMPARISONS OF THE TREE-MODEL AND THE TABLE METHOD IN THE MOUNTAIN CAR SIMULATION**

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of Leaf Nodes</th>
<th>Total nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree-Model with T-Statistics</td>
<td>56.4</td>
<td>111.8</td>
</tr>
<tr>
<td>Tree-Model with Weighted T-Statistics</td>
<td>52.325</td>
<td>105.65</td>
</tr>
<tr>
<td>Table model</td>
<td>X</td>
<td>1089</td>
</tr>
</tbody>
</table>

Dyna-Q takes until the 49th episodes. Therefore, the proposed method can accelerate the speed of learning.

Table IV compares the tree-model and the table model. The tree-model uses different methods, T-statistics and weighted T-statistics to reduce the number of leaf nodes. The split method by weighted T-statistics can use fewer leaf nodes to approximate the mountain car model. The policy learning, Q-learning, has the same resolution when the state space is uniformly partitioned into 2000 blocks with a size of 50 × 40. By comparison, the table model is uniformly partitioned into 1089 blocks with a size of 33 × 33, so it consumes more memory than the tree-model. The tree-model is built by a decision tree, so the number of leaf nodes and internal nodes need to be considered. The total count of nodes is two times of number of leaf nodes minus one.

**B. Searching Goal Problem**

This simulation includes three mobile robots that have different starting points, as shown in Fig. 17. The size of the maze is 300 × 300 with walls. The environment can be divided into three areas, with a robot in each area. The three square frames, labeled by “R1,” “R2,” and “R3,” are starting points of three mobile robots. The goal is labeled by “G.” In the simulation, Robot 2 can approach the goal easily, but Robot 1 and Robot 3 have difficulty in getting to the goal by crossing the small paths. Robot 1 needs to explore Area 1 more exhaustively as well as Robot 3 should do in Area 3. The action set of all agents includes four actions, moving up, down, left, or right. The moving distance of each action is 5, and all actions have added Gaussian noise, which uses the moving distance as mean, and the variance is set to 0.5. The obstacles in the maze are indicated by black rectangles. After the agents take an action, there are three levels of reward, −100, +100 and −0.00001. If the agent touches the goal, it receives the reward of +100 and if it hits a wall, it receives the reward of −100. For other actions, the agent receives a reward of −0.00001. In the learning process, when the learning steps reach 2000 or the agent touches the goal, an episode is terminated, and the agent runs 500 episodes per round. The parameter settings in the problem of searching for the goal are listed in Table V.

In this simulation, Robot 2 can share knowledge with Robot 1 and Robot 3, and this knowledge sharing helps them decrease the explorative efforts and approach the goal sooner. In Figs. 18 and 19, the y-axis is the average number of steps and the x-axis is the number of episodes. According to the figures, the sharing methods can reduce the exploration time to search for goal. The SuC and SRR have the approximate performance in Robot 1 and Robot 3. However, the SuC has the over-sharing problem, which is caused by redundant sharing samples being sent by other agents repeatedly and persistently;
even the receiver already has had sufficient local knowledge. Such phenomenon obviously influences the performance of Robot 1.

In this simulation, the environment is not complex, so the agents can use fewer average number of leaf nodes to build the world model according to Table VI. But the agents share the knowledge voluntarily in the sharing method of SuC, which means the agents require many leaf nodes to build the world model. The SRR method took the fewest average leaf nodes.

The resultant partitions for the model of Robot 1, depicted in Fig. 20(a)–(d), shows the outcomes produced by the methods of no-sharing, SuC, SRUS, SRR, respectively. Robot 1 in Area 1 needs to cross a narrow path to enter Area 2, so its model partitions in open areas are coarse but become finer around walls and goal, due to the design of sparse reward functions in these simulation cases. Comparison of the performance and the number of the leaf nodes demonstrate that the sharing method of SRR is the best sharing method. But since the environment is not complex; the difference of the number of leaf nodes is not obvious. The next simulation increases the complexity of the environment and compares the differences in performance and in the number of leaf nodes.

C. Mobile Robots in Maze

This simulation increases the complexity based on the searching goal problem which includes three mobile Robots that have different starting points, as shown in Fig. 21. The size of the maze is $300 \times 300$ with walls. The three square frames labeled R1, R2, and R3, are starting points of the mobile Robots. The goal is labeled G in the center of the maze. In the simulation, Robot 3 is the farthest from the goal and Robot 1 is the closest. The robots’ task is to collect experiences to build their own model and to find a path to approach the goals simultaneously. In the process of building the model, they share their model information and the other agents use the information to speed up the model learning. The obstacles in the maze are black U-shaped blocks and the black circle. The settings of action space and the reward values are the same as the searching goal problem. In the learning process, when the learning steps reach 3000 or an agent touches the goal, an episode is terminated. Each agent runs 500 episodes per round. The settings of parameters for mobile robots in the maze problem are listed in Table VII.

As depicted in Figs. 22 and 23, the y-axis is the average number of steps and the x-axis is the number of episodes.
Because Robot 1 begins near the goal, the average number of steps cannot express the efficiency. The averaged numbers of steps for Robot 2 and Robot 3 are compared in the figures. The results show the average number of steps to get to the goal of each episode, with four curves for no-sharing and the proposed sharing methods. The proposed methods can be more efficient than the model without the knowledge sharing. In Fig. 22, the proposed sharing methods can make Robot 2 approach the goal faster than that with no-sharing, but the figure cannot clearly show the difference of performance between the three sharing methods. The steps of Robot 3 are shown in Fig. 23, and the SuC method can arrive at the goal in an average of 500 steps in the 87th episodes, the SRUS method can reach the goal in an average of 500 steps in the 136th episodes and the SRR method achieve the goal in the 75th episodes, and the no-sharing method achieves the goal until the 162th episode. Therefore, the proposed sharing methods can speed up learning.

The numbers of leaf nodes for the three robots are listed in Table VIII. The SRR method has the least average number of leaf nodes, and the SuC method produces more number of leaf nodes than no-sharing. Fig. 24 shows the models partitioned in Robot 3 by the four methods. Fig. 24(a) is the result generated by no-sharing. Fig. 24(b), the result of the SuC method, shows that it partitioned more leaf nodes than other methods. When an agent terminates an episode, it shares its knowledge with other agents. This sharing method makes agents share knowledge repeatedly, and the over-sharing problem arises. Fig. 24(c) sketches the result of the SRUS method, which makes many threadlike splits. The result of the SRR method is shown in Fig. 24(d). It can decrease the possibility of threadlike splits and produce minimal number of leaf nodes, as listed in Table VIII.

**VI. Conclusion**

In Dyna-Q, the same RL method is used both for learning from real experiences and for planning from virtual experiences. Learning and planning in Dyna-Q are deeply integrated in the sense that they share almost all the same mechanism, except for the source of their experiences. Conceptually, planning, acting, model-learning, and direct RL occur simultaneously and in parallel in Dyna agents. For implementation on a serial computer, these processes run in sequence within a time step and require little computation, and consume only a fraction of the time. The remaining time slot can be devoted to planning, indirect RL, which is inherently computation-intensive.

In the maze examples, models in the agents started empty, and were then filled with experiences. Whereas, models with sparse information may be inaccurate, especially in the case of a vast environment where most states only have a limited number of samples or are never visited. When the model is
incorrect, the planning process may go astray. Therefore, the knowledge sharing mechanism is proposed to enforce model-learning process that only spends time mostly on data communication, resampling, and tree induction. That obviously costs less computation time than the one in planning.

The proposed tree-model method can extend easily to the domain of multiagent learning with sharing. This paper studied three successive knowledge sharing methods for model construction between agents who are learning a task with a vast state space such that a tremendously computational time must be spent in exploring the entire state space before taking actions in the optimal way given the current model. The sharing methods take advantage of local experiences gained by each agent exploring around its vicinity. Since the environmental model held by each agent is represented by a decision tree. Sharing occurs by means of grafting the leaf nodes with sufficient experiences to the one unfamiliar with the area. Therefore, agents can always save the efforts in exploring areas that have been frequently visited by other agents already.

In this paper, the proposed Dyna-Q architecture with a tree-model and three successively improved sharing methods have been applied to a multiagent system. In the proposed method, agents share out their dedicated local experiences to help a demanding partner produce a more global environment model. Learning by real robots in real time became possible. Since the domain of multiagent learning with sharing. This paper studied three successive knowledge sharing methods for model construction between agents who are learning a task with a vast state space such that a tremendously computational time must be spent in exploring the entire state space before taking actions in the optimal way given the current model. The sharing methods take advantage of local experiences gained by each agent exploring around its vicinity. Since the environmental model held by each agent is represented by a decision tree. Sharing occurs by means of grafting the leaf nodes with sufficient experiences to the one unfamiliar with the area. Therefore, agents can always save the efforts in exploring areas that have been frequently visited by other agents already.

In this paper, the proposed Dyna-Q architecture with a tree-model and three successively improved sharing methods have been applied to a multiagent system. In the proposed method, agents share out their dedicated local experiences to help a demanding partner produce a more global environment model. By utilizing the model to learn, cost to learn has been reduced such that learning by real robots in real time became possible. To demonstrate the effectiveness of the proposed method, simulations of two labyrinths and a mountain car have been carried out. As a result, the learning time has been reduced and effective memory consumption has been acquired.

REFERENCES


Kao-Shing Hwang (M’93–SM’09) received the M.M.E. and Ph.D. degrees in electrical and computer engineering from Northwestern University, Evanston, IL, USA, in 1989 and 1993, respectively. He is a Professor with the Electrical Engineering Department, National Sun Yat-sen University, Kaohsiung, Taiwan, and an Adjunct Professor with the Electrical Engineering Department, Chung Cheng University, Chia-Yi, Taiwan. Since 1993, he has been with National Chung Cheng University. He was the Chairman of the Electrical Engineering Department from 2003 to 2006, and the Chairman of Optimechatronics Institute, National Chung Cheng University from 2010 to 2011. He is also in charge of establishing the standardized exchange platform of heterogeneous educational resources under the Ministry of Education of Taiwan. His current research interests include methodologies and analysis for various intelligent robot systems, machine learning, embedded system design, and ASIC design for robotic applications.

Wei-Cheng Jiang was born in Taiwan in 1985. He received the B.S. degree in computer science and information engineering and the M.S. degree in electro-optical and materials science from National Formosa University, Yunlin County, Taiwan. He is currently pursuing the Ph.D. degree from the Department of Electric Engineering, National Chung Cheng University, Chia-Yi, Taiwan. His current research interests include neural networks, learning systems, fuzzy control, and mobile robots.

Yu-Jen Chen received the B.S. degree in electrical engineering from the Tatung Institute of Technology, Taipei, Taiwan, in 1994, and the M.S. and Ph.D. degrees in electrical engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 1997 and 2009, respectively. From 2004 to 2009, he was an Adjunct Lecturer with the Center for General Education, National Chung Cheng University. Since 2010, he has been an Assistant Professor with the Electrical Engineering Department, National Chung Cheng University. His current research interests include machine learning, robotics, neural networks, and embedded systems.