Model Learning and Knowledge Sharing for Cooperative Multiagent Systems in Stochastic Environment

Wei-Cheng Jiang, Vignesh Narayanan, *Member, IEEE*, and Jr-Shin Li, *Senior Member, IEEE*

***Abstract*—An imposing task for a reinforcement learning agent in an uncertain environment is to expeditiously learn a policy or a sequence of actions, with which it can achieve the desired goal. In this article, we present an incremental model learning scheme to reconstruct the model of a stochastic environment. In the proposed learning scheme, we introduce a clustering algorithm to assimilate the model information and estimate the probability for each state transition. In addition, utilizing the reconstructed model, we present an experience replay strategy to create virtual interactive experiences by incorporating a balance between exploration and exploitation, which greatly accelerates learning and enables planning. Furthermore, we extend the proposed learning scheme for a multiagent framework to decrease the effort required for exploration and to reduce the learning time in a large environment. In this multiagent framework, we introduce a knowledge-sharing algorithm to share the reconstructed model information among the different agents, as needed, and develop a computationally efficient knowledge fusing mechanism to fuse the knowledge acquired using the agents' own experience with the knowledge received from its teammates. Finally, the simulation results with comparative analysis are provided to demonstrate the efficacy of the proposed methods in the complex learning tasks.**

***Index Terms*—Knowledge sharing, model learning, multiagent system, reinforcement learning (RL), sample efficiency.**

I. Introduction

REINFORCEMENT learning (RL) is a popular machine-learning tool employed to tackle complex decision-making problems in uncertain environments. Inspired by naturalistic decision-making processes in biological systems, the RL methodology enables an agent to progressively learn a sequence of actions to achieve a desired objective [1]. The environment, wherein the agent performs an action, provides a scalar feedback, called the reinforcement/reward signal, and as a consequence of the action, the agent transits from one state (current state) to a new state. During this process of interaction with the environment, the agent explores the environment and accumulates experiences or knowledge regarding the environment. Using the collected experiences, over time, the agent learns the best sequence of action from any situation/state [2], [3] to achieve a desired objective.

Typically, these experiences are collected when the agent is exploring the environment, that is, the agent actually takes actions in an uncertain environment hoping that these actions do not jeopardize the agent and receives reinforcement signals and collects experiences. Therefore, the process of collecting experiences in an uncertain environment is expensive and time consuming, and it is imperative that the collected experiences are efficiently utilized [4]–[7]. For example, a rover exploring an uncertain terrain may crash onto an obstacle or plunge into a crevasse. In this context, sample efficiency in RL is a challenging and important issue, that is, extracting the model/policy information efficiently from the collected experience becomes an important objective.

The RL approaches are broadly classified as model-free and model-based learning strategies. In the model-free RL, the collected experiences are used to learn a policy based on the expected value of applying an action at each step. These experiences are discarded once the learning is complete. On the other hand, the model-based RL approach is introduced to achieve sample efficiency [8]–[10]. In the model-based RL approach, the agent strives to reconstruct a virtual model of the environment which not only includes the reward function but also the transition function, the function that encodes how an action from the current state is mapped to the next state.

The Dyna architecture is a well-known model-based RL approach [11], [12] in which an agent reconstructs a model of the environment from actual experience (through direct learning), and after the learning process, when the model is fed with the current state and an action, it will predict the next state and the expected reward. This allows for the agent to create virtual experiences that can be used to update the policy and incorporate planning efficiently through indirect learning.

Manuscript received July 15, 2019; revised September 30, 2019; accepted November 26, 2019. The work of W.-C. Jiang was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 106-2917-I-564-028 and Grant MOST 108-2218-E-029-006. The work of J.-S. Li was supported in part by NSF under Award ECCS-1509342 and Award CMMI-1763070, and in part by the National Institutes of Health under Grant R01GM131403 and Grant R01CA226937A1. This article was recommended by Associate Editor H. M. Schwartz. *(Corresponding author: Jr-Shin Li.)*

W.-C. Jiang is with the Department of Electrical Engineering, Tunghai University, Taichung 40704, Taiwan, and also with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA.

V. Narayanan and J.-S. Li are with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (e-mail: jsli@wustl.edu).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2019.2958912

Indirect learning also enables the agent to plan and achieves a better balance between exploration and exploitation [13]. Several algorithms that improve/accelerate the learning process in the model-based RL framework, have been proposed in [14]–[16].

An alternate approach to achieve sample efficiency in the learning process is to employ experience replay strategy [17]–[20]. In both the Dyna architecture and the experience replay strategy, all the collected experiences are stored. However, in some applications, the memory space to store the experiences may be limited or costly. Therefore, in the experience replay strategy, a mechanism to omit experiences during the learning process is introduced. Different algorithms incorporating SARSA, Q-learning, etc., are studied along with the experience replay strategies to improve the learning efficiency [21]. The characteristic difference of the traditional Q-learning or SARSA learning with the experience replay-based strategies is the additional mechanism which discards experiences when the memory is full. This mechanism works either based on time (discarding oldest experience first) or based on the reward (discarding experiences with least reward) [20].

It should be noted that the RL algorithms, for example, Q-learning, propagate the Q-values from the terminal state to the initial state during the learning process. As a result, experiences collected initially, that are near the initial states, may be useful, and when some experiences are omitted based on time or reward, these useful experiences may be discarded. This renders the learning process inefficient. Similarly, in the Dyna architecture, the agent, during indirect learning, always chooses an action from a state that is visited previously to predict the next state and the corresponding reward. In some cases, the environment (state space) is too large for an agent to cover, slowing down this learning process.

Therefore, when a single agent deals with a large environment, the learning time will increase. In such a case, a suitable learning scheme, incorporating multiple learning agents that cooperate and share their experiences collected independently with the other agents [22]–[25], will accelerate the learning process. The multiagent learning framework can accommodate a variety of decision-making tasks [26], for instance, tasks where multiple agents cooperate or compete to maximize a coupled objective (e.g., Half-field offence [27]). However, in this article, we will adopt the multiagent learning framework to accelerate the learning process by using a team of RL agents to learn a large environment. Here, each agent will take actions that are independent of the other agents and share their knowledge with the other agents when there is a request for information. Such cooperative knowledge-sharing methods were reported in [22], [23], and [28]–[32], and the references therein. For instance, the agents explored the environment independently to update their Q-functions and then shared the Q-values with their teammates as needed in the results presented in [28]–[30]. In these approaches, each agent's policy is shared with the others [28]–[32]. Alternatively, in a model sharing approach, each agent builds its own model of the environment during the learning period and exchanges its model information with the other agents [33]–[37].

In the cooperative knowledge-sharing approach, each agents' policy/model is shared with the others, and an important task for an agent is to effectively combine the newly obtained knowledge from the other agents with the knowledge gained using its own experience [28]–[32]. This depends on how the information is stored in each agent. In [32], the heterogeneous tree structures were proposed, for each agent, for state aggregation, where the Q-values were stored in different tree nodes. A model-learning scheme using heterogeneous tree structures along with a request-based knowledge-sharing framework was proposed by Hwang *et al.* [37]. In contrast to the algorithm in [38], where the entire tree structure was shared with the other agents, in [37], agents only shared their leaf nodes to the others to reduce the merging cost. However, due to the heterogeneous tree structure, a resampling method was introduced to create virtual experiences for tractable fusing of the knowledge shared between the cooperative agents.

In summary, sample efficiency and reduction in the learning time in a large, uncertain environment could potentially be resolved if an algorithm can: 1) efficiently reconstruct a virtual model that can store more information, especially, in a stochastic state space; 2) generate useful experiences from the virtual model for indirect learning, without discarding potentially valuable experiences; and 3) effectively share knowledge in the multiagent setting and combine the shared knowledge without increasing the computational burden on each agent.

Therefore, in this article, we propose a multiagent learning scheme, wherein each agent collects experiences for policy and model learning in an uncertain, stochastic environment. To learn the environmental model information from the experience collected by each agent, we introduce an incremental model learning algorithm, and to effectively capture the state transition, we propose a clustering scheme. In the proposed clustering-based model learning approach, clusters are formed to store the information regarding the probability for each state transition. As opposed to the traditional "look-up" table-based storage of the environmental models or the tree structure, the clusters offer a concise way to store the model information.

Furthermore, we present an experience replay strategy along with the Q-learning scheme to achieve sample efficiency, by incorporating a balance between exploration and exploitation. The experience replay strategy is used to create virtual experiences corresponding to states which may or may not be experienced by the agents, for updating the Q-function. Furthermore, when an agent in a cooperative team lacks sufficient knowledge of a specific state, a request will prompt its teammates to share their model information. After obtaining the knowledge, an efficient fusing mechanism (using the clusters and T-statistics) is proposed to help each agent fuse its own knowledge and the knowledge shared by its teammates. The efficiency of the proposed algorithms is validated with three numerical simulations, and the comparison with related state-of-the-art methods demonstrates the advantages of the proposed scheme.

The contributions of this article include developing: 1) an incremental model learning algorithm to establish a virtual model of a stochastic environment; 2) a clustering scheme to assimilate the experiences in a virtual model to evaluate

the probability of the state transitions; 3) an experience replay strategy with incremental model learning schemes to achieve sampling efficiency via multistep Q-learning; and 4) a knowledge fusing scheme for the multiagent framework to fuse the model knowledge shared by the teammates.

The remainder of this article is organized as follows. In Section II, a brief background on RL is presented and the problem considered in this article is detailed. The proposed incremental model learning method along with the clustering scheme are presented in Section III. Section IV introduces the knowledge-sharing algorithm which is employed in tandem with the incremental model learning method in a multiagent setting. In Section V, the simulation results for the proposed method are presented. Finally, in Section VI, we present our conclusions.

II. Background

In this section, we first provide a brief background on the RL problem and then state the design objectives considered in this article. The RL problems can be formulated as Markov decision processes (MDPs) which consist of a state space $\boldsymbol{S}_p$, an action space $\boldsymbol{A}_p$, a state-transition function $\boldsymbol{T}$, and a reward function $\boldsymbol{R}$ [1]. During the learning process, an RL agent perceives the environmental information, denoted as a state $\boldsymbol{s}_t \in \boldsymbol{S}_p$ and chooses an action $a_t \in \boldsymbol{A}_p$ so that the agent transits from the current state to the next state $\boldsymbol{s}_{t+1}$ based on $\boldsymbol{T}$. The action a_t is, in general, based on a policy, and in this process, the agent receives an external scalar reward/reinforcement signal $r_{t+1} \in \boldsymbol{R}$ from the environment. The tuple $(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}, r_{t+1})$ will be collected as an experience by the RL agent and the aggregated experiences will be used to update the policy, which will influence the agent's future actions. Specifically, the RL agent uses the experiences to update the long-term expected reward that is stored as the value function or the Q-function.

In the RL tasks, the model of the environment can be deterministic or stochastic [1], [39]. In the deterministic environment, the transition function $\boldsymbol{T} : \boldsymbol{S}_p \times \boldsymbol{A}_p \rightarrow \boldsymbol{S}_p$ maps a state and an action to a new next state in the state space, while in the stochastic environment, the transition function maps the state and the action to the next state with a probability for each state transition defined by a distribution. Therefore, in a stochastic environment, the RL agent must encode the probability distribution using the collected experiences in the learnt model. In this article, we consider an MDP with a continuous state space $\boldsymbol{X}_p \subseteq \mathbb{R}^n$, a discrete action space $\boldsymbol{A}_p$, a stochastic state-transition function $\boldsymbol{T}_p$, and a reward function $\boldsymbol{R}_p$. In addition, we define a map $\boldsymbol{s}_t \leftarrow \phi(\boldsymbol{x}_t)$, that maps the continuous states $\boldsymbol{x}_t$ to a discrete state $\boldsymbol{s}_t \in \boldsymbol{S}_p \subseteq \mathbb{R}^n$.

Furthermore, learning with multiple RL agents enables solving complex tasks in a large state space [26] and can include a wide variety of RL problems (e.g., Nash games). In this article, we focus on developing an efficient model learning and knowledge-sharing mechanism in a large stochastic environment by using multiple agents. Specifically, we let the agents learn a model of the environment, and a policy to reach a goal, while also helping their teammates by sharing their model and policy when requested.

The objectives considered in this article are: 1) to develop a model learning scheme with efficient storage structure to facilitate indirect learning; 2) to design an algorithm for incorporating indirect learning/planning for the RL agent to efficiently utilize the virtual model; 3) to design a multiagent knowledge-sharing framework; and 4) to develop a knowledge fusing algorithm without imparting computational burden (by avoiding a resampling scheme) on each agent.

III. Model Learning for Experience Replay With Q-Learning

In this section, an incremental model learning scheme for experience replay is presented. Since the major focus of this article is not Q-learning, for completeness, the details of the policy learning scheme are briefly introduced first.

A. Policy Learning via Direct Learning

At time t, an agent perceives a state $\boldsymbol{x}_t = [x_1, x_2, \ldots, x_n]$, where n is the dimension of the state space. The perceived continuous state is encoded to a discretized state $\boldsymbol{s}_t = [s_1, s_2, \ldots, s_n] \in \boldsymbol{S}_p$ using the map ϕ. An ϵ-greedy method is used by the agent to choose an action $a_t \in \boldsymbol{A}_p$ in the current state $\boldsymbol{s}_t$ [1]. Following the state transition, the agent will perceive the next continuous state $\boldsymbol{x}_{t+1}$, which is encoded to the next discretized state $\boldsymbol{s}_{t+1}$, and a reward r_{t+1} is returned by the environment. In the model-free Q-learning approach, the agent utilizes the experience $(\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}, r_{t+1})$ to update the Q-values denoted as $Q(\boldsymbol{s}_t, a_t)$. The Q-function [40] is given by

$$Q(\boldsymbol{s}_t, a_t)_{\text{new}} = (1-\beta)Q(\boldsymbol{s}_t, a_t)_{\text{old}} + \beta\left(r_{t+1} + \gamma \max_{a' \in \boldsymbol{A}_p} Q(\boldsymbol{s}_{t+1}, a')\right) \quad (1)$$

where $0 < \beta < 1$ is the learning rate and $0 < \gamma < 1$ is the discount factor. If the agent acquires a desirable reward, the Q-values will be increased for the corresponding state–action pair, while the Q-values will be decreased, otherwise. With the Q functions updated iteratively, the Q-values for all state–action pairs will converge to the optimal Q^* values asymptotically with probability one [41], [42]. The update error is defined as

$$\Delta Q(\boldsymbol{s}_t, a_t) = Q(\boldsymbol{s}_t, a_t)_{\text{new}} - Q(\boldsymbol{s}_t, a_t)_{\text{old}}. \quad (2)$$

In our proposed algorithm, the agent learns the optimal policy via both the direct learning and indirect learning, where $Q(\boldsymbol{s}, a)$ stores the Q-values for all the state–action pairs, $\Delta Q(\boldsymbol{s}, a)$ stores all the updating error, and $M(\boldsymbol{s}, a)$ stores the state-transition and reward functions. Next, we present our incremental model learning scheme for encoding the model information in $M(\boldsymbol{s}, a)$ using the collected experiences.

B. Incremental Model Learning via Direct Learning

In addition to learning the Q-function, the experiences $(\boldsymbol{x}_t, a_t, \boldsymbol{x}_{t+1}, r_{t+1})$ are utilized for learning the model of the environment. First, the model learning scheme is presented for the deterministic environment and, then, for the stochastic environment.

1) Deterministic Environment: For a deterministic environment, if the agent takes an action a_1 from the continuous state $\boldsymbol{x}_1$, it will transit to the next continuous state $\boldsymbol{x}_2$ with probability 1.

In this article, we propose a model that is composed of two functions, a variation transition function $\boldsymbol{T}_v : \boldsymbol{S}_p \times \boldsymbol{A}_p \rightarrow \Delta\boldsymbol{X}_p \subseteq \mathbb{R}^n$, which maps a state and an action to a variation of state (or the first difference), and a reward function $\boldsymbol{R}_v : \boldsymbol{S}_p \times \boldsymbol{A}_p \times \boldsymbol{S}_p \rightarrow \mathbb{R}$, which maps a state and an action to a scalar reward. For instance, the Nth state transition due to an action a results in the variation of the continuous state $\Delta\boldsymbol{x}_N^a$ and the reward r_N^a that is computed as

$$\Delta\boldsymbol{x}_N^a = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t, \quad r_N^a = r_{t+1}. \tag{3}$$

The variation of the continuous state and the reward is stored in a model, in cells, wherein each cell stores the variation of the continuous state $\Delta\boldsymbol{x}_1^a$, the square of the variation of the continuous state $(\Delta\boldsymbol{x}_1^a)'$, the reward r_1^a, and the square of the reward $(r_1^a)'$. If the agent visits a state more than once, say $(N-1)$ times, the variations and rewards are averaged and stored. These quantities are calculated as

$$\begin{aligned}
\Delta\bar{\boldsymbol{x}}_N^a &= \frac{\Delta\boldsymbol{x}_1^a + \Delta\boldsymbol{x}_2^a + \cdots + \Delta\boldsymbol{x}_{N-1}^a}{N-1} \\
\left(\Delta\bar{\boldsymbol{x}}_N^a\right)' &= \frac{\left(\Delta\boldsymbol{x}_1^a\right)^2 + \left(\Delta\boldsymbol{x}_2^a\right)^2 + \cdots + \left(\Delta\boldsymbol{x}_{N-1}^a\right)^2}{N-1} \\
\bar{r}_N^a &= \frac{r_1^a + r_2^a + \cdots + r_{N-1}^a}{N-1} \\
\left(\bar{r}_N^a\right)' &= \frac{\left(r_1^a\right)^2 + \left(r_2^a\right)^2 + \cdots + \left(r_{N-1}^a\right)^2}{N-1}
\end{aligned} \tag{4}$$

where $\Delta\bar{\boldsymbol{x}}_N^a$ is the averaged variation of the continuous state, $(\Delta\bar{\boldsymbol{x}}_N^a)'$ is the average of squared variation of the continuous state, $\bar{r}_N^a$ is the average of the reward, and $(\bar{r}_N^a)'$ is the average of squared reward.

Hence, at the Nth visit to this state, the new $\Delta\bar{\boldsymbol{x}}_{N+1}^a$ and $(\Delta\bar{\boldsymbol{x}}_{N+1}^a)'$ are updated according to

$$\begin{aligned}
\Delta\bar{\boldsymbol{x}}_{N+1}^a &= \frac{1}{N}\sum_{i=1}^{N}\Delta\boldsymbol{x}_i^a = \frac{1}{N}\left(\Delta\boldsymbol{x}_N^a + \sum_{i=1}^{N-1}\Delta\boldsymbol{x}_i^a\right) \\
&= \frac{1}{N}\left(\Delta\boldsymbol{x}_N^a + (N-1)\frac{1}{(N-1)}\sum_{i=1}^{N-1}\Delta\boldsymbol{x}_i^a\right) \\
&= \Delta\bar{\boldsymbol{x}}_N^a + \frac{1}{N}\left(\Delta\boldsymbol{x}_N^a - \Delta\bar{\boldsymbol{x}}_N^a\right)
\end{aligned} \tag{5}$$

$$\left(\Delta\bar{\boldsymbol{x}}_{N+1}^a\right)' = \left(\Delta\bar{\boldsymbol{x}}_N^a\right)^2 + \frac{1}{N}\left(\left(\Delta\boldsymbol{x}_N^a\right)^2 - \left(\Delta\bar{\boldsymbol{x}}_N^a\right)^2\right). \tag{6}$$

Since it is infeasible to know *a priori* if a state will be visited by the agent more than once, (4) is not implementable and the incremental update in (6) can be used.

Analogously, $\bar{r}_{N+1}^a$ and the new $(\bar{r}_{N+1}^a)'$ are updated by

$$\bar{r}_{N+1}^a = \bar{r}_N^a + \frac{1}{N}\left(r_N^a - \bar{r}_N^a\right) \tag{7}$$

$$\left(\bar{r}_{N+1}^a\right)' = \left(\bar{r}_N^a\right)^2 + \frac{1}{N}\left(\left(r_N^a\right)^2 - \left(\bar{r}_N^a\right)^2\right). \tag{8}$$

Algorithm 1 Incremental Model Learning for Deterministic Environment (**ModelLearning**)

(1) Calculate the variation of the continuous state and the reward using equation (3)
(2) $\boldsymbol{s} \leftarrow \phi(\boldsymbol{x}_t)$
(3) Retrieve $\Delta\bar{\boldsymbol{x}}_N^a$, $(\Delta\bar{\boldsymbol{x}}_N^a)'$, $\bar{r}_N^a$, and $(\bar{r}_N^a)'$, from the model using state- action pair, $(\boldsymbol{s}, a)$
(4) Use (5)-(8) to calculate the new $\Delta\bar{\boldsymbol{x}}_{N+1}^a$, $(\Delta\bar{\boldsymbol{x}}_{N+1}^a)'$, $\bar{r}_{N+1}^a$ and $(\bar{r}_{N+1}^a)'$
(5) Update $\boldsymbol{T}_v(\Delta\bar{\boldsymbol{x}}_{N+1}^a|\boldsymbol{s}, a)$ and $\boldsymbol{R}_v(\boldsymbol{s}', \boldsymbol{s}, a)$ using (9)
(6) Store these two functions in the model,

$$M(\boldsymbol{s}, a) \leftarrow (\boldsymbol{T}_v(\Delta\bar{\boldsymbol{x}}_{N+1}^a|\boldsymbol{s}, a), \boldsymbol{R}_v(\boldsymbol{s}', \boldsymbol{s}, a))$$

(7) Return $M(\boldsymbol{s}, a)$.

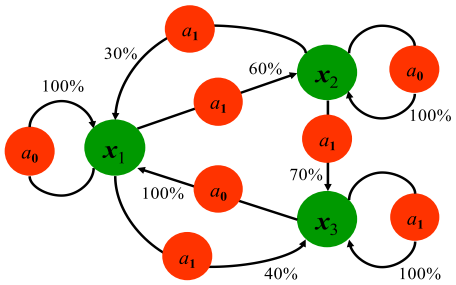

Fig. 1. Diagram representing a three-state Markov process, with the states labeled x_1, x_2, and x_3 and actions a_0 and a_1. Each number represents the probability of the Markov process changing from one state to another state, with the direction indicated by the arrow.

Therefore, the variation transition function and the reward function are handled by the proposed model that is defined as

$$\begin{aligned}
\boldsymbol{T}_v\left(\Delta\bar{\boldsymbol{x}}_{N+1}^a|\boldsymbol{s}, a\right) &= \left\{\Delta\bar{\boldsymbol{x}}_{N+1}^a|\boldsymbol{s} \leftarrow \phi(\boldsymbol{x}_t), a \leftarrow a_t\right\} \\
\boldsymbol{R}_v\left(\boldsymbol{s}', \boldsymbol{s}, a\right) &= \left\{\bar{r}_{n+1}^a|\left(\boldsymbol{s}', \boldsymbol{s}\right) \leftarrow \phi(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t), a \leftarrow a_t\right\}
\end{aligned} \tag{9}$$

where $\boldsymbol{T}_v$ maps the discretized state $\boldsymbol{s}$ and the action a_t to an average variation of the continuous states $\Delta\bar{\boldsymbol{x}}_{N+1}^a \in \Delta\boldsymbol{X}_p$, and the reward function $\boldsymbol{R}_v$ maps $\boldsymbol{s}'$ and $\boldsymbol{s}$, and the action a to an average reward $\bar{r}_{N+1}^a \in \mathbb{R}$. The algorithm of the incremental model learning method for a deterministic environment is shown in Algorithm 1.

2) Stochastic Environment: For the case of stochastic environment, the probability distribution for the state transitions needs to be taken into consideration. In the example shown in Fig. 1, when an agent in the continuous state $\boldsymbol{x}_1$ chooses an action a_1, it would transit to the next continuous state $\boldsymbol{x}_2$ or $\boldsymbol{x}_3$ with probabilities 0.6 and 0.4, respectively. If the agent in the continuous state $\boldsymbol{x}_2$ takes the action a_1, it will have different transition probabilities for the next continuous states. Since the probability distribution of the state transitions is unknown, in the model-based RL, the probability distribution must be encoded in the environmental model that is being learned by the agent.

Therefore, for the stochastic environment, the variation transition function becomes $\boldsymbol{T}_p : \boldsymbol{S}_p \times \boldsymbol{A}_p \rightarrow \mathbb{P}(\Delta\boldsymbol{X}_p)$, where $\Delta\boldsymbol{X}_p \subseteq \mathbb{R}^n$ is the variation of the state space and $\mathbb{P}$ is the probability operator and, similarly, the reward function is modified as $\boldsymbol{R}_p : \boldsymbol{S}_p \times \boldsymbol{A}_p \times \boldsymbol{S}_p \rightarrow \mathbb{P}(\boldsymbol{R}_p)$, where $\boldsymbol{R}_p \subseteq \mathbb{R}$ is the

reward value. To evaluate and encode the probability distribution in the model, we introduce a clustering method in addition to the incremental model learning scheme proposed for the deterministic case. The clustering algorithm evaluates the variation vector of the state, and the reward values for each state transition and assigns them to different clusters.

Specifically, when an agent obtains an experience $(\boldsymbol{x}_t, a_t, \boldsymbol{x}_{t+1}, r_{t+1})$, (3) is used to calculate the normalized variation of the state to obtain the normalized variation vector $\boldsymbol{v} = (\Delta \boldsymbol{x}^a_G, r^a_G)$. The normalization is carried out as follows. Let $d_{i,m}$ and $d_{i,M}$ denote the lower and upper bounds of the ith dimension of the state space, and let R_m and R_M denote the smallest and largest reward. Then, the normalized state variation $(\Delta \boldsymbol{x}^a_G)$ and normalized reward (r^a_G) are calculated as

$$\Delta x^a_{G,i} = \frac{\Delta x^a_{N,i} - d_{i,m}}{d_{i,M} - d_{i,m}}, \quad r^a_G = \frac{r^a_N - R_m}{R_M - R_m} \tag{10}$$

where $i = 1, 2, \ldots, |\Delta x^a_{N,i}| = n$, $\Delta x^a_{N,i} \in \Delta \boldsymbol{x}^a_N$ and $\Delta x^a_{G,i} \in \Delta \boldsymbol{x}^a_G$.

When the first normalized variation vector is received, a new cluster will be established and it will be set as the center of the cluster $\boldsymbol{c}_1 = \boldsymbol{v}$, and the information of this cluster is updated using (5)–(8). When the next normalized variation vector $\boldsymbol{v}'$ arrives, the Euclidean distance is used to calculate the similarity, $D(\boldsymbol{v}', \boldsymbol{c}_l)$, between the new normalized variation vector $\boldsymbol{v}'$ and the centers of the existing clusters $\boldsymbol{c}_l$, $l = 1, 2, \ldots, |C|$, where $|C|$ is the cardinality of the cluster set, $C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_l\}$.

Next, the most similar cluster $\underline{\boldsymbol{c}_l}$ will be chosen and the similarity $d_{\underline{\boldsymbol{c}_l}}$ is compared with a threshold D^a_{th}. If the similarity is smaller than the threshold, the cluster $\underline{\boldsymbol{c}_l}$ is activated and the information of this cluster is updated by (5)–(8). The number of times a cluster is activated is represented by N_{c_l}. Therefore, in each discretized cell of the model, there may exist multiple clusters, and the probabilities of each cluster in a cell are evaluated using the activation number. The probabilities are used in the experience replay strategy to create a collection of virtual experiences. Therefore, in addition to the $M(\boldsymbol{s}, a)$, in the stochastic environment, the cluster centers and the activation number of each cluster are stored in the model.

In Fig. 2, an example cell and the model structure $M(\boldsymbol{s}, a)$ are shown for an environment with 1-D state and action spaces. The x, y-axes represent the actions and continuous state, discretized into cell blocks. Each discrete cell may contain several clusters, for instance, c_1, c_2, and c_3 are the center of the three clusters in a cell. In each of these clusters, the activation time N_{c_1} of cluster 1 is 8, N_{c_2} is 8, and N_{c_3} is 6. The algorithm of the incremental model learning method for stochastic environment is shown in Algorithm 2.

Note that the proposed model learning algorithms incorporate a discretization step and, then, store state variation, rewards, and their respective squared values (10). However, in contrast to the deterministic model, the stochastic model must encode the transition probabilities associated with each state transition. In both the cases, a finer discretization can provide a better approximation of the learned model, it increases the computations and, thus, introduces a tradeoff between model accuracy and computational complexity. On the other hand,

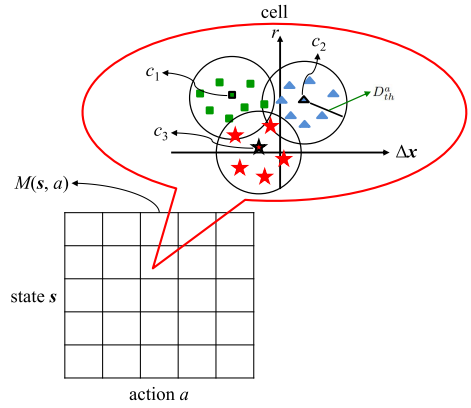

Fig. 2. Depiction of information stored in the model learned by an agent. The model structure $M(\boldsymbol{s}, a)$ corresponds to an environment with 1-D state and action spaces. The x, y-axes represent the actions and continuous state, discretized into cell blocks (black square boxes). Each discrete cell may contain several clusters, for instance (the red pop-out) shows the contents of a cell. This cell contains three clusters wherein c_1, c_2, and c_3 are their centers, respectively. In this example, the activation time N_{c_1} for cluster 1 is 8, N_{c_2} is 8, and N_{c_3} is 6.

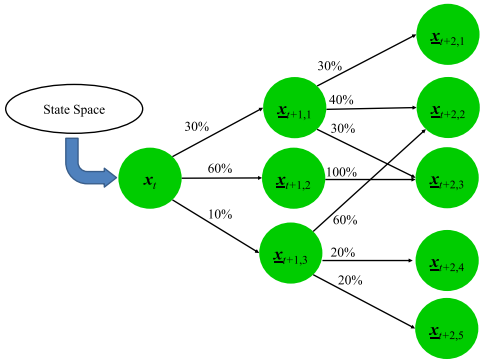

Fig. 3. Example of multistep planning.

the number of clusters in the stochastic environment is not predetermined, and agents automatically vary the number of clusters based on the complexity of the environment.

Next, an indirect learning algorithm incorporating experience replay is proposed to improve the sample-efficiency of the RL scheme.

C. Experience Replay for Planning (Indirect Learning)

In planning, virtual experiences, created using the model reconstructed from actual experiences via direct learning, are employed to update the Q-function (and as a consequence, the policy). To increase the efficiency of planning, we propose a multistep planning algorithm as illustrated in Fig. 3.

To generate virtual experiences, first, a continuous state $\underline{\boldsymbol{x}_t}$ is discretized as $\underline{\boldsymbol{s}_t}$ and an action $\underline{a_t}$ must be chosen from the action space. Different from the Dyna architecture, in this article, the action is chosen from the action space such that

$$\underline{a_t} = \arg\max_{a' \in \boldsymbol{A}_p} \left((1 - \lambda) e^{\Delta Q(\underline{\boldsymbol{s}_t}, a')} + (\lambda) \kappa \right) \tag{11}$$

where $\lambda \in [0, 1]$ is a constant and κ is a random number selected in the interval of $[0, 1]$. The action selection criterion

Algorithm 2 Incremental Model Learning for Stochastic Environment (**ModelLearning**)

(1) Calculate the variation of the continuous state and the reward using equation (3)
(2) A normalized procedure is employed to calculated a normalized variation vector, $\boldsymbol{v} \leftarrow (\Delta \boldsymbol{x}_N^a, r_N^a)$
(3) $\boldsymbol{s} \leftarrow \phi(\boldsymbol{x}_t)$
(4) Retrieve all clusters, $C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_l\}$, from the cell of the model, $M(\boldsymbol{s}, a)$ using state-action pair, $(\boldsymbol{s}, a)$, $|C|$ is the total number of cluster in this cell.
(5) **if** $|C| == 0$ **then**
(6) The first variation vector is set as the center of the first cluster, i.e., $\boldsymbol{c}_1 = \boldsymbol{v}$, $\boldsymbol{c}_1 \in C$
(7) **else if** $|C| > 0$ **then**
(8) $d_{\boldsymbol{c}_{\underline{l}}} \leftarrow arg\min_{\boldsymbol{c}_l \in C} D(\boldsymbol{c}_l, \boldsymbol{v})$
(9) **if** $d_{\boldsymbol{c}_{\underline{l}}} > D_{th}^a$ **then**
(10) Create a new cluster, $\boldsymbol{c}_{l+1} = \boldsymbol{v}, \boldsymbol{c}_{l+1} \in C$
(11) **else**
(12) Activate the cluster, $\boldsymbol{c}_{\underline{l}}$, and retrieve the information from this cluster, $\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}}$, $(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}})'$, $(\bar{r}^a_{N_{c_{\underline{l}}}})$ and $(\bar{r}^a_{N_{c_{\underline{l}}}})'$
(13) Calculate the new $\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}}$, the new $(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}})'$, the new $(\bar{r}^a_{N_{c_{\underline{l}}+1}})$ and the new $(\bar{r}^a_{N_{c_{\underline{l}}+1}})'$ using the equations (5)-(8).
(14) $N_{c_{\underline{l}}} \leftarrow N_{c_{\underline{l}}} + 1$
(15) **end if**
(16) **end if**
(17) Update the variation transition function, $\boldsymbol{T}_p(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}}|\boldsymbol{s}, a)$, and the reward function, $\boldsymbol{R}_p(\boldsymbol{s}', \boldsymbol{s}, a)$

$$\boldsymbol{T}_p(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}}|\boldsymbol{s}, a) \leftarrow \Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}}$$
$$\boldsymbol{R}_p(\boldsymbol{s}', \boldsymbol{s}, a) \leftarrow \bar{r}^a_{N_{c_{\underline{l}}+1}}$$

(18) Store these two functions in the model, $M(\boldsymbol{s}, a)$.

$$M(\boldsymbol{s}, a) \leftarrow (\boldsymbol{T}_p(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}+1}}|\boldsymbol{s}, a), \boldsymbol{R}_p(\boldsymbol{s}', \boldsymbol{s}, a))$$

(19) return $M(\boldsymbol{s}, a)$

evaluates the actions from the given state based on their updating error and picks the action with a large error. Alternatively, when all the actions have minor and comparable updating errors, an action is randomly picked. The first part of this selection rule enables exploitation, while the second part is used to incorporate exploration. If λ is selected close to zero, then the agent tends to choose an action based on the current Q values (with less preference to explore) while if λ is selected close to 1, the agent tends to randomly choose an action (promoting exploration) (see Fig. 10 in the supplementary material).

After choosing an action, the discretized state $\underline{\boldsymbol{s}}_t$ and an action $\underline{a}_t$ are the input to the model. This state–action pair $(\underline{\boldsymbol{s}}_t, \underline{a}_t)$ will activate a cell of the model, which may contain several clusters. Using the activation number of clusters in the cell, the probability for each cluster is defined as

$$P_{c_l}(\underline{\boldsymbol{s}}_t, \underline{a}_t) = \frac{N_{c_l}}{\sum_{l=1}^{|C|} N_{c_l}}. \tag{12}$$

The cluster with higher probability, $\underline{c_l}$, is selected to retrieve the variation vector and expected reward to form the virtual

Algorithm 3 Experience Replay for Stochastic Environment (**ExperienceReplay**)

(1) $g \leftarrow 0$
(2) Choose a simulated continuous state, $\underline{\boldsymbol{x}}_t$, from the state-space
(3) **while** $g < K$ **do**
(4) $\underline{\boldsymbol{s}}_t \leftarrow \phi(\underline{\boldsymbol{x}}_t)$
(5) $\underline{a}_t \leftarrow arg\max_{a' \in \boldsymbol{A}_p}((1-\lambda)e^{\Delta Q(\underline{\boldsymbol{s}}_t, a')} + (\lambda)\kappa)$
(6) $(\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}}, \bar{\boldsymbol{r}}^a_{N_{c_{\underline{l}}}}) \leftarrow M(\underline{\boldsymbol{s}}_t, \underline{a}_t)$
(7) $\underline{\boldsymbol{x}}_{t+1} \leftarrow \underline{\boldsymbol{x}}_t + \Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}}$
(8) $\underline{\boldsymbol{s}}_{t+1} \leftarrow \phi(\underline{\boldsymbol{x}}_{t+1})$
(9) Use (1) and (2) to update the Q-value and ΔQ-value
(10) $\underline{\boldsymbol{x}}_t \leftarrow \underline{\boldsymbol{x}}_{t+1}$
(11) $g \leftarrow g + 1$
(12) **end while**

experience. For example, in Fig. 2, the probability of clusters 1–3 approximates 36.36%, 36.36%, and 27.27%, respectively.

From the activated cluster $\underline{c_l}$, the average variation of the continuous state $\Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}}$ and the average reward $\bar{r}^a_{N_{c_{\underline{l}}}}$ are retrieved, and the next continuous state is calculated as

$$\underline{\boldsymbol{x}}_{t+1} = \underline{\boldsymbol{x}}_t + \Delta\bar{\boldsymbol{x}}^a_{N_{c_{\underline{l}}}}. \tag{13}$$

The virtual experience, $(\underline{\boldsymbol{x}}_t, \underline{a}_t, \underline{\boldsymbol{x}}_{t+1}, \bar{r}^a_{N_{c_{\underline{l}}}})$, is discretized and is used to update the policy. The algorithm of experience replay is listed in Algorithm 3, where K is the number of the steps in the multistep planning algorithm. The Q-learning has been proven to converge after the agents have had many interactions with the environment [41]. As the number of interactions between the agent and the environment increases, it is observed that the model-learning algorithms learn a reliable representation of the environment [12]. However, global convergence of the learning algorithms requires sufficient exploration of the environment and finer discretization. In the proposed indirect learning algorithm, the parameter λ controls the exploration, and it aids in the learning process (see Fig. 10 in the supplementary material). Furthermore, note that the number of clusters is not predefined in our algorithm, and the agent forms clusters online. This online clustering process influences the learning wherein the number of clusters depends on the complexity of the environment, and as the complexity increases, the agents will store the model information using more clusters automatically.

Note that it is possible for an RL agent to perform indirect learning using a model that may not be fully accurate. In the proposed algorithm, the model information includes (12), which indicates that as the number of visits increases, using the virtual experiences from the cluster with higher number of visits can be very useful. If the number of visits to a cluster is low, the agent requests information from its teammates and uses a knowledge-sharing scheme (see Section IV). This reduces the possibility of an agent trying to improve its Q-table over an imprecise environment. The agent despite using the model for indirect learning, continues to refine the model by using actual experiences via direct learning.

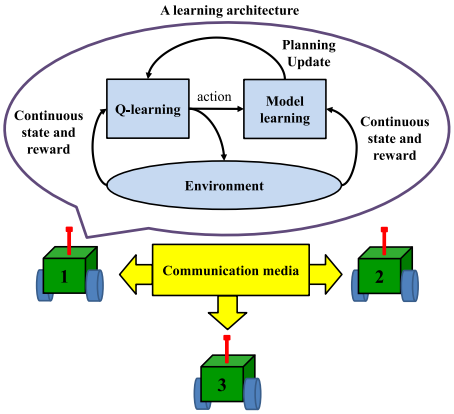

Fig. 4. Sharing architecture for multiple agents.

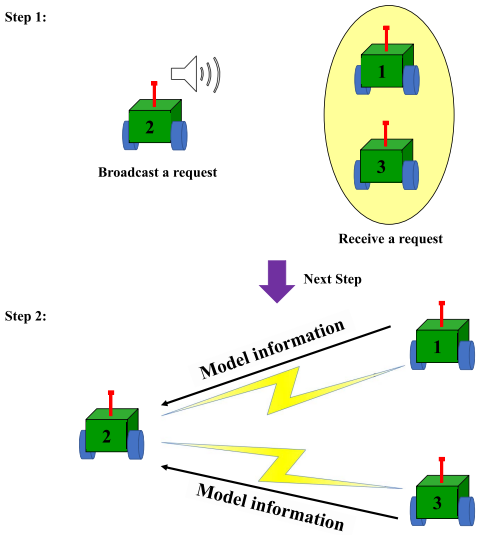

Fig. 5. Illustration of the sharing process.

Next, the proposed model learning scheme for the RL agent and the experience replay algorithm are extended to a more general case involving multiagent system.

IV. Knowledge Sharing in Multiagent System

A sharing architecture for multiple agents is shown in Fig. 4. In the multiagent architecture, m homogeneous agents have their own Q-table for policy learning and their own model for model learning. The cooperative learning process is separated into two modes: 1) a learning mode and 2) a sharing mode. In the learning mode, an agent must collect experiences that are used for policy learning and model learning. In a learning task, especially, in a large state space, there may be many areas in the state space unvisited by any agent.

For example, when an agent obtains a continuous state $\underline{\boldsymbol{x}}_{t+1,1}$ and an action, it tries to use the model to predict the next continuous state and the reward. However, in some situations, the cell of the model has insufficient information, namely, it has not been visited by the agent or the visiting times of the cell V are lower than a prescribed threshold V_{th}, where the visiting time is defined by

$$V = \sum_{l=1}^{|C|} N_l \tag{14}$$

with $|C|$ being the number of clusters that is stored in the cell of the model.

If V is lower than V_{th}, the cell is viewed as one with insufficient information. In such a case, to compensate for the insufficient model information, this agent must switch to the sharing mode and the process of information exchange between agents is shown in Fig. 5. In this example, first, agent 2 with insufficient information broadcasts a request including the continuous state for which the information is needed. Upon receiving the request, agents 1 and 3 find the corresponding cell in their models and send the corresponding model information back to agent 2 in the second step. Agent 2 then fuses its original model information with the received model information from the other agents. Here, we introduce a fusing mechanism to achieve this goal. Let $C_{sh} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_q]$ be the clusters received by an agent from its teammates and let $C_{or} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_o]$ be its own clusters corresponding to that cell with the respective cardinality denoted as $|C_{sh}|$ and $|C_{or}|$. Before the fusing procedure, the variance of the variation of the continuous states $(\underline{s}_s^a)^2$ and the variance of the reward $(\underline{s}_r^a)^2$ for all the clusters in the sets, $|C_{sh}|$ and $|C_{or}|$, are calculated as

$$\left(\underline{s}_s^a\right)^2 = \left(\Delta\bar{\boldsymbol{x}}_N^a\right)' - \left(\Delta\bar{\boldsymbol{x}}_N^a\right)^2,\ \left(\underline{s}_r^a\right)^2 = \left(\bar{r}_N^a\right)' - \left(\bar{r}_N^a\right)^2. \tag{15}$$

Next, to initiate the fusing procedure, T-statistic [43] is used wherein the similarity between any two clusters is quantified based on a t-value. For each cluster pair (one from C_{or} and the other from C_{sh}), a t-value vector composed of $\boldsymbol{t}_v = [t_1, t_2, \ldots, t_n]$, where n is the dimension of the state space, and t_r corresponding to the reward is calculated as

$$\boldsymbol{t}_v = \frac{\left|\Delta\bar{\boldsymbol{x}}_{N,or}^a - \Delta\bar{\boldsymbol{x}}_{N,sh}^a\right|}{\sqrt{\left(\frac{\left(\underline{s}_{s,or}^a\right)^2}{N_{i,or}}\right) + \left(\frac{\left(\underline{s}_{s,sh}^a\right)^2}{N_{l,sh}}\right)}} \tag{16}$$

$$t_r = \frac{\left|\bar{r}_{N,or}^a - \bar{r}_{N,sh}^a\right|}{\sqrt{\left(\frac{\left(\underline{s}_{r,or}^a\right)^2}{N_{i,or}}\right) + \left(\frac{\left(\underline{s}_{r,sh}^a\right)^2}{N_{l,sh}}\right)}} \tag{17}$$

where N_i and N_l are the activation number of the ith, lth cluster from C_{or} and C_{sh}, respectively.

Note that in the proposed algorithm, the agent receives clusters C_{sh} from all its teammates. During the implementation of this algorithm, the agent, upon requesting information from its teammates, will receive clusters from all its teammates, which are stored in a buffer. Then, the agent will retrieve these clusters from the buffer and will update the model in batches. Also, if the agents do not have sufficient information (determined based on V and V_{th}), no new information will be sent to the agent which initiated a request, and the buffer will be empty.

If one element of the t-value vector $\boldsymbol{t}_n$ and t_r are larger than the threshold t_{th}, then, these two clusters will not be merged into one cluster. Otherwise, if all the elements of $\boldsymbol{t}_n$ and t_r

are smaller than the threshold, it implies that the two clusters have similar sample distributions and they can be merged into one cluster. In such a case, the information of this cluster is updated. In particular, the average of the variation of the continuous state and of the reward will be updated as

$$\begin{aligned}\Delta\bar{\boldsymbol{x}}^a_{N_{\text{total,or}}} &= \frac{N_{i,or}}{N_{\text{total}}} \times \Delta\bar{\boldsymbol{x}}^a_{N,or} + \frac{N_{l,sh}}{N_{\text{total}}} \times \Delta\bar{\boldsymbol{x}}^a_{N,sh} \\ \bar{r}^a_{N_{\text{total,or}}} &= \frac{N_{i,or}}{N_{\text{total}}} \times \bar{r}^a_{N,or} + \frac{N_{l,sh}}{N_{\text{total}}} \times \bar{r}^a_{N,sh} \end{aligned} \tag{18}$$

and the new average of the squared variation of the continuous state and the reward will be updated as

$$\begin{aligned}\left(\Delta\bar{\boldsymbol{x}}^a_{N,or}\right)' &= \frac{N_{i,or}}{N_{\text{total}}} \times \left(\Delta\bar{\boldsymbol{x}}^a_{N,or}\right)' + \frac{N_{i,sh}}{N_{\text{total}}} \times \left(\Delta\bar{\boldsymbol{x}}^a_{N,sh}\right)' \\ \left(\bar{r}^a_{N,or}\right)' &= \frac{N_{i,or}}{N_{\text{total}}} \times \left(\bar{r}^a_{N,or}\right)' + \frac{N_{l,sh}}{N_{\text{total}}} \times \left(\bar{r}^a_{N,sh}\right)' \end{aligned} \tag{19}$$

where $N_{\text{total}} = N_{i,or} + N_{l,sh}$. We observe that the weights ($[N_{i,or}/N_{\text{total}}]$, $[N_{i,sh}/N_{\text{total}}]$) in these update rules can be chosen between $(0, 1)$. In our simulations, we update the model information using the convex combination of the agents own knowledge and the knowledge shared by the teammates [as in (18) and (19)]. Thus, by using the clustering approach, the introduced fusing mechanism circumvents the resampling scheme required in the existing algorithm [37]. This is achieved because of the cluster-based model learning, which avoids the heterogeneous tree structures.

Next, simulation analysis for the proposed methods is presented.

V. Simulations

In this section, first, a mountain car simulation is used to compare the learning efficiency for a single agent. The proposed method is compared with Q-learning [41], backward Q-learning (BQ-learning) [21], Q-learning with experience replay based on reward and time [QExp-learning (Time) and QExp-learning (Reward)] [20], and Dyna-Q learning [12].

Next, we evaluate the framework where multiple agents learn the state-space model and a policy. To this end, we apply the proposed knowledge-sharing algorithm to the multiagent system studied in the goal search problem and in a maze environment and compare the performance resulting from our method with other commonly used sharing methods in [33] and [37]. Here, we do not consider tasks involving co-operative or competitive games. Therefore, the agents choose their actions independently. In all these examples, the learning task is complete when each robot in the team learns to reach the goal. We do assume that there is no loss due to communication during knowledge sharing. In Examples 2 and 3, we analyze the knowledge-sharing algorithm, and we present a comparative analysis between the proposed methods and some existing methods, in terms of the learning steps. Since each algorithms use different learning structure to approximate the environment, the information stored in each algorithm is different, and the balance between exploration and exploitation by each agent in the environment is different among the algorithms. Therefore, the comparison between algorithms in terms of communication instances would not be fair. However, the termination criterion for each episode is fixed in the comparative analysis with different methods and, therefore, the convergence of the algorithms in terms of the episodes implicitly provide a qualitative comparison on the time for convergence of the learning algorithms.

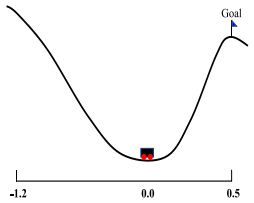

Fig. 6. Mountain car simulation.

To evaluate the statistical significance of the results, we perform a Monte-Carlo analysis and record the average number of steps required for convergence of the algorithm (over multiple episodes) and its standard deviation. For each example, we include the average convergence and comparative analysis and present detailed implementation procedure in the main text, and the standard deviation of the number of steps required for convergence over multiple episodes are recorded in the supplementary material.

A. Mountain Car

In the mountain car simulation (Fig. 6), a car in the valley has three actions, that is, throttle forward $(+1)$, throttle reverse (-1), or zero throttle (0). However, these actions cannot output sufficient power to drive the car to reach the hill. As a result, the car must learn a policy to save the potential energy by applying a sequence of actions so that it can be propelled to reach the goal. The dynamics of the mountain car are

$$\begin{aligned} p_{t+1} &= p_t + \dot{p}_{t+1}, \\ \dot{p}_{t+1} &= \dot{p}_t + 0.01a_t + 0.01\omega_t - 0.0025\cos(3.0 \times p_t) \end{aligned} \tag{20}$$

where $p_t \in [-1.2, 0.5]$ is the position of the car, $\dot{p}_t \in [-0.07, 0.07]$ is the velocity of the car, and ω_t is a Gaussian random noise process [1]. In the process of learning, if the car reached the goal, an immediate reward of $+1.0$ was returned by the environment. Otherwise, the car received a reward of -1.0. There were 30 trials for the simulation, and each trail included 200 episodes with 1500 steps per episode. If the car reached the hill or it cannot reach the hill within 1500 steps, then the episode was terminated, and a new episode was started. In each episode, the initial position and velocity of the car were set to -0.5 and 0.0, respectively. The setting of the parameters are listed in Table I in the supplementary material. The resolution of the Q-table and the proposed model were selected as 50×40 discretized grids.

The average numbers of the steps for the car to reach the goal are shown in Fig. 7. The x-axis represents the number of episodes and the y-axis represents the number of steps. In this figure, six curves represent the results from Q-learning, BQ-learning, Q-learning with experience replay based on time and reward, Dyna-Q, and the proposed methods (Algorithms 2 and 3), respectively. In this comparison shown in Fig. 7, the

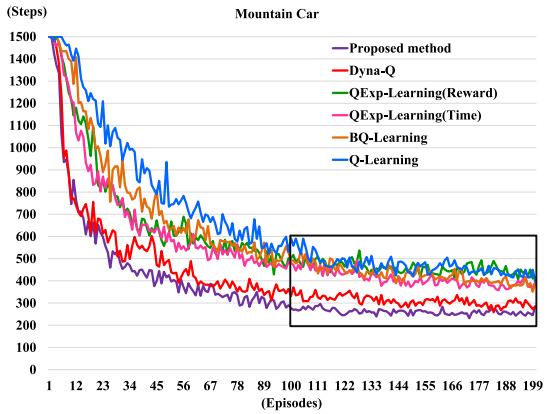

Fig. 7. Simulation results in mountain car.

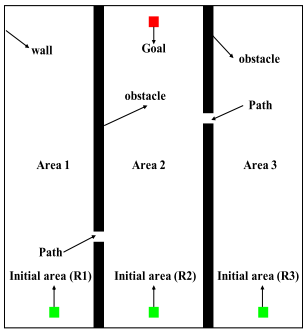

Fig. 8. Cooperative goal search.

proposed method reduced the average steps required to reach the goal to be lower than 300 and accelerated the learning process in comparison with the existing methods.

B. Cooperative Goal Searching Problem

In this example, three cooperative robots are placed in the environment as shown in Fig. 8. The size of the environment is 300×300 and surrounded by walls, with obstacles. The three robots are placed in different areas labeled as Area 1, Area 2, and Area 3. Robot 1 (R1), robot 2 (R2), and robot 3 (R3) start from their respective initial areas R1, R2, and R3 (painted with green color). The size of these initial areas is 10×10 units. If a robot finishes an episode, it will randomly start from its initial position (the area marked in green) in the next episode. The red area is the goal.

In this simulation, robot 2 is the nearest to the goal, and hence it is easier for it to reach the goal, while robots 1 and 3 need to explore more exhaustively to find a path to avoid the obstacle and eventually reach the goal. Therefore, in this cooperative task, robot 2 obtains useful knowledge easily, and when the robots 1 and 3 enter R2 (Area 2), the robot 2 can share its own knowledge to help the other robots.

In this task, all the three robots have the same action space, {moving up, down, left, and right}. The moving distance for each action is set as 5 in the presence of the Gaussian noise with zero mean and standard deviation 0.5. After the robots take an action, they met with three types of situations: 1) if a robot hits the wall or the obstacle, it received a reward of

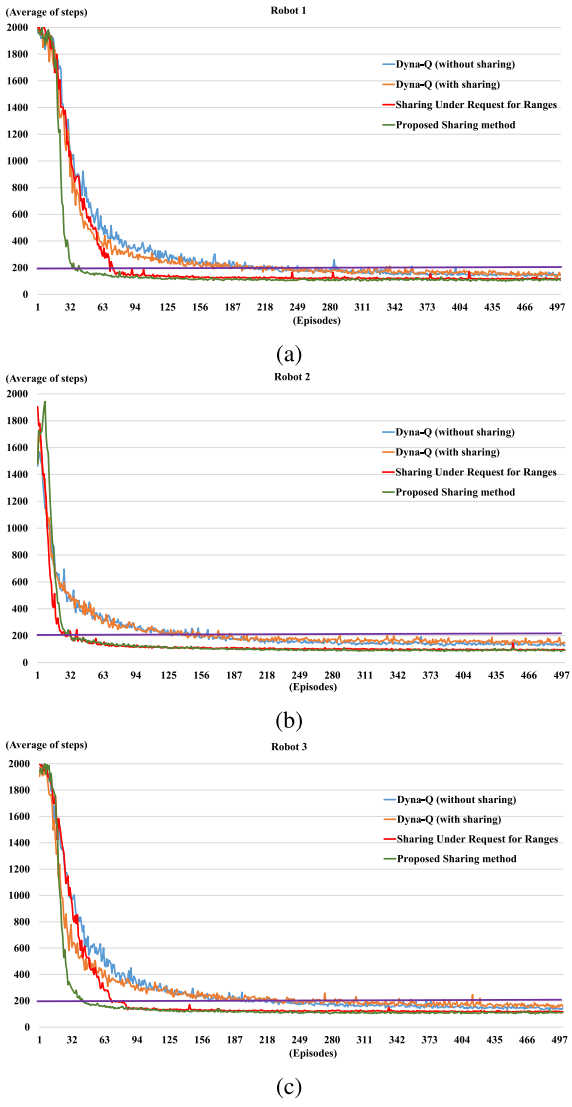

Fig. 9. Simulation results in cooperative searching goal problem. (a) Learning curves for robot 1. (b) Learning curves for robot 2. (c) Learning curves for robot 3.

-100.0; 2) if a robot reached the goal, it received a reward of 100.0; and 3) else, it received a reward of -10^{-6}.

In the process of learning, when the number of steps reached 2000 or if the robot touched the goal, an episode was ended. There were 500 episodes per trial and 40 trials. The parameters for the cooperative searching goal problem are listed in Table II in the supplementary material. The resolution of the Q-table and the proposed model was selected as 100×100 discretized grids. Using the proposed sharing mechanism, robots 1 and 3 will decrease the required exploration by using the knowledge gained by robot 2. The proposed method is compared with some other knowledge-sharing methods. In the algorithm presented by Tan [33], all the robots shared their experiences with each other, while in [37], three sharing mechanisms based on a tree structure were proposed. Based on the simulation results, the sharing method, sharing under request for ranges, increased the learning efficiency considerably in comparison with the other schemes.

In Fig. 9, the x-axis is the number of episodes and the y-axis is the average number of steps. In Fig. 9(a), the proposed method decreased the average steps required for robot 1 to reach the goal down to 200 in the 36th episode. The sharing under request decreased the average steps to 200 in the 70th episode. Dyna-Q with sharing experiences decreased the average steps in 169th episode, and the Dyna-Q without sharing experiences decreased the average steps until the 196th episode.

Since robot 2 was near the goal, the difference of the average steps was not noticeable as shown in Fig. 9(b). The learning curve of robot 3 is shown in Fig. 9(c), where it can be observed that the proposed method decreased the average steps for robot 3 to reach the goal to 200 in the 43rd episode. The sharing under request decreased the average steps down to 200 in the 70th episode while the Dyna-Q with sharing experiences decreased the average steps in the 195th episode, and the Dyna-Q without sharing experiences in the 197th episode.

According to these simulation results, the Dyna-Q with sharing experiences reduced the average steps required for convergence in the initial episodes, but not in the later episodes, while the sharing under request for ranges decreased the average steps slower than the Dyna-Q with sharing in the initial episodes, but as the learning episodes increased, as a team, the multiagent framework improved the knowledge gathered, and as a consequence, the convergence was improved in the later episodes. On the other hand, with the proposed method, the average steps required both in the initial and the later episodes decreased. The third simulation example and additional analyses are presented in the Supplementary Material.

VI. Conclusion

In this article, a model-based RL scheme was proposed wherein the collected experiences are not only used for policy learning but also used for model learning to approximate the environment (deterministic and stochastic). By employing the learnt model, the agent simulated virtual experiences for indirect learning and planning. The clustering method introduced in this article was employed in tandem with the proposed incremental model learning method to evaluate the transition probability and encode this in the model. The simulation results and comparative analysis demonstrated that the proposed methods require fewer learning steps to reach the goal and as a consequence improve the sample efficiency.

Moreover, to solve the complex learning tasks in a large state space, the proposed methods we extended to incorporate knowledge sharing in a multiagent system. In other words, the cost of exploration can be reduced by leveraging the benefits of knowledge sharing and the simulation results indicated that the learning steps are considerably reduced with the proposed method. Furthermore, due to the clustering method employed in the model learning, the knowledge fusing procedure circumvents the computationally intensive resampling methods.

In this article, we did not consider problems with cooperative/competitive games, where actions of an agent affects other agents. These problems require learning joint state and joint action spaces and are practically significant, and the proposed algorithms can be expanded to incorporate such tasks. The second line of inquiry would involve theoretical analysis of the proposed learning algorithms and, finally, explicitly accounting for the communication losses, such as delay, data drop out, noisy data in the knowledge-sharing scheme, and circumventing the discretization step would further expand the application of the proposed algorithms.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] K. Iwata, "Extending the peak bandwidth of parameters for softmax selection in reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 8, pp. 1865–1877, Aug. 2017.

[3] X. Xu, Z. Huang, D. Graves, and W. Pedrycz, "A clustering-based graph Laplacian framework for value function approximation in reinforcement learning," *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2613–2625, Dec. 2014.

[4] M. Leonetti, L. Iocchi, and P. Stone, "A synthesis of automated planning and reinforcement learning for efficient, robust decision-making," *Artif. Intell.*, vol. 241, pp. 103–130, Dec. 2016.

[5] T. Hester and P. Stone, "Generalized model learning for reinforcement learning in factored domains," in *Proc. 8th Int. Conf. Auton. Agents Multiagent Syst.*, vol. 2, 2009, pp. 717–724.

[6] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2010, pp. 2369–2374.

[7] A. M. Farahmand, A. Shademan, M. Jagersand, and C. Szepesvári, "Model-based and model-free reinforcement learning for visual servoing," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2009, pp. 2917–2924.

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

[9] H. H. Viet, P. H. Kyaw, and T. Chung, "Simulation-based evaluations of reinforcement learning algorithms for autonomous mobile robot path planning," in *IT Convergence and Services*. Amsterdam, The Netherlands: Springer, 2011, pp. 467–476.

[10] T. Hester and P. Stone, "An empirical comparison of abstraction in models of Markov decision processes," in *Proc. ICML/UAI/COLT Workshop Abstraction Reinforcement Learn.*, 2009, pp. 18–23.

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

[12] L. Kuvayev and R. S. Sutton, "Model-based reinforcement learning with an approximate, learned model," in *Proc. 9th Yale Workshop Adapt. Learn. Syst.*, 1996, pp. 1–13.

[13] H. Viet, S. An, and T. Chung, "Extended Dyna-$\mathcal{Q}$ algorithm for path planning of mobile robots," *J. Meas. Sci. Instrum.*, vol. 2, no. 3, pp. 283–287, 2011.

[14] H. Shi, S. Yang, K.-S. Hwang, J. Chen, M. Hu, and H. Zhang, "A sample aggregation approach to experiences replay of Dyna-$\mathcal{Q}$ learning," *IEEE Access*, vol. 6, pp. 37173–37184, 2018.

[15] W. Caarls and E. Schuitema, "Parallel online temporal difference learning for motor control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 7, pp. 1457–1468, Jul. 2016.

[16] R. Kamalapurkar, L. Andrews, P. Walters, and W. E. Dixon, "Model-based reinforcement learning for infinite-horizon approximate optimal tracking," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 753–758, Mar. 2017.

[17] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 293–321, 1992.

[18] S. Kalyanakrishnan and P. Stone, "Batch reinforcement learning in a complex domain," in *Proc. ACM 6th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2007, p. 94.

[19] F. Ruelens *et al.*, "Residential demand response of thermostatically controlled loads using batch reinforcement learning," *IEEE Trans. Smart Grid*, vol. 8, no. 5, pp. 2149–2159, Sep. 2017.

[20] M. Pieters and M. A. Wiering, "$\mathcal{Q}$-learning with experience replay in a dynamic environment," in *Proc. SSCI*, 2016, pp. 1–8.

[21] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward Q-learning: The combination of Sarsa algorithm and Q-learning," *Eng. Appl. Artif. Intell.*, vol. 26, no. 9, pp. 2184–2193, 2013.
[22] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, "FMRQ—A multiagent reinforcement learning algorithm for fully cooperative tasks," *IEEE Trans. Cybern.*, vol. 47, no. 6, pp. 1367–1379, Jun. 2017.
[23] F. L. Da Silva, R. Glatt, and A. H. R. Costa, "MOO-MDP: An object-oriented representation for cooperative multiagent reinforcement learning," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 567–579, Feb. 2019.
[24] F. L. da Silva, R. Glatt, and A. H. R. Costa, "An advising framework for multiagent reinforcement learning systems," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 4913–4914.
[25] F. L. Da Silva, R. Glatt, and A. H. R. Costa, "Simultaneously learning and advising in multiagent reinforcement learning," in *Proc. 16th Conf. Auton. Agents MultiAgent Syst.*, 2017, pp. 1100–1108.
[26] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.
[27] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for RoboCup soccer keepaway," *Adapt. Behav.*, vol. 13, no. 3, pp. 165–188, 2005.
[28] M. N. Ahmadabadi, M. Asadpour, and E. Nakano, "Cooperative Q-learning: The knowledge sharing issue," *Adv. Robot.*, vol. 15, no. 8, pp. 815–832, 2001.
[29] M. N. Ahmadabadi and M. Asadpour, "Expertness based cooperative Q-learning," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 1, pp. 66–76, Feb. 2002.
[30] B. N. Araabi, S. Mastoureshgh, and M. N. Ahmadabadi, "A study on expertise of agents and its effects on cooperative Q-learning," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 2, pp. 398–409, Apr. 2007.
[31] B. H. Abed-Alguni, "Bat Q-learning algorithm," *Jordan. J. Comput. Inf. Technol.*, vol. 3, no. 1, pp. 56–77, 2017.
[32] Y.-J. Chen, K.-S. Hwang, and W.-C. Jiang, "Policy sharing between multiple mobile robots using decision trees," *Inf. Sci.*, vol. 234, pp. 112–120, Jun. 2013.
[33] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 330–337.
[34] K. Ito, A. Gofuku, Y. Imoto, and M. Takeshita, "A study of reinforcement learning with knowledge sharing for distributed autonomous system," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom.*, vol. 3, 2003, pp. 1120–1125.
[35] K. Ito, Y. Imoto, H. Taguchi, and A. Gofuku, "A study of reinforcement learning with knowledge sharing-applications to real mobile robots," in *Proc. IEEE Int. Conf. Robot. Biomimet. (ROBIO)*, 2004, pp. 175–180.
[36] T. Tateyama, S. Kawata, and T. Shimomura, "Parallel reinforcement learning systems using exploration agents and Dyna-Q algorithm," in *Proc. IEEE Annu. Conf. (SICE)*, 2007, pp. 2774–2778.
[37] K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, "Model learning and knowledge sharing for a multiagent system with Dyna-Q learning," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 978–990, May 2015.
[38] M. Asadpour, M. N. Ahmadabadi, and R. Siegwart, "Heterogeneous and hierarchical cooperative learning via combining decision trees," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2006, pp. 2684–2690.
[39] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, Apr. 1996.
[40] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Dept. Psychol., King's College, Cambridge, U.K., 1989.
[41] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[42] Q. Wei, F. L. Lewis, Q. Sun, P. Yan, and R. Song, "Discrete-time deterministic Q-learning: A novel convergence analysis," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1224–1237, May 2017.
[43] L. D. Pyeatt and A. E. Howe, "Decision tree function approximation in reinforcement learning," in *Proc. 3rd Int. Symp. Adapt. Syst. Evol. Comput. Probab. Graph. Models*, vol. 2, 2001, pp. 70–77.

Wei-Cheng Jiang 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, Taiwan, in 2007 and 2009, respectively, and the Ph.D. degree in electric engineering from National Chung Cheng University, Chiayi, Taiwan, in 2013.

He was a Postdoctoral Fellow with the Electrical Engineering Department, National Sun Yat-sen University, Kaohsiung, Taiwan. He was a Visiting Scholar with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, USA. Since 2019, he has been an Assistant Professor with the Department of Electrical Engineering, Tunghai University, Taichung, Taiwan. His research interests include machine learning, multiagent systems, and intelligent control.

Vignesh Narayanan (M'17) received the B.Tech. degree from SASTRA University, Thanjavur, India, the M.Tech. degree from the National Institute of Technology Kurukshetra, Kurukshetra, India, and the Ph.D. degree from the Missouri University of Science and Technology, Rolla, MO, USA, in 2017.

He is currently working as a Postdoctoral Research Associate with Washington University in St. Louis, St. Louis, MO, USA. His research interests include control, neural networks, learning, and adaptation in systems theory.

Jr-Shin Li (M'06–SM'18) received the B.S. and M.S. degrees from National Taiwan University, Taipei, Taiwan, and the Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA, USA, in 2006.

He is currently a Professor of electrical and systems engineering with the Joint Appointment Division of Biology and Biomedical Sciences, Washington University in St. Louis, St. Louis, MO, USA. His research interests are control theory, computational mathematics, optimization, learning, and complex networks. His current work involves developing model-based and data-driven methods for dynamical systems and control of large-scale complex systems with applications ranging from neuroscience and biology to quantum physics.

Prof. Li has been a recipient of the NSF Career Award in 2007 and the AFOSR Young Investigator Award in 2009.