International Journal of Innovative Computing, Information and Control
Volume **18**, Number **2**, April **2022**

ICIC International ©2022 ISSN 1349-4198
pp. **447–462**

COMBINATIONS OF MICRO-MACRO STATES AND SUBGOALS DISCOVERY IN HIERARCHICAL REINFORCEMENT LEARNING FOR PATH FINDING

Gembong Edhi Setyawan[1], Hideyuki Sawada[1] and Pitoyo Hartono[2]

[1]Department of Applied Physics
School of Advanced Science and Engineering
Waseda University
3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
gembong@asagi.waseda.jp; sawada@waseda.jp

[2]School of Engineering
Chukyo University
101-2 Yagoto Honmachi, Showa-ku, Nagoya, Aichi 466-8666, Japan
hartono@sist.chukyo-u.ac.jp

Received September 2021; revised January 2022

Abstract. *While Reinforcement Learning (RL) is one of the strongest unsupervised learning algorithms, it often faces difficulties dealing with complex environments. These difficulties correlate with the curse of dimensionality in which an excessively large number of states causes the process of RL prohibitively difficult. Hierarchical Reinforcement Learning (HRL) is proposed to overcome the weaknesses of RL by hierarchically decomposing a complex problem into more manageable sub-problems. This paper proposes Micro-Macro States Combination (MMSC) as a new approach for HRL by formulating the task into two layers. The lower layer depicts the task in their microstates, which represent the original states, while the upper layer depicts macrostates, some collections of a number of the microstates. The macrostates can be considered the higher abstractions of the original states that allow the RL to perceive the problem differently. Here, the proposed MMSC is allowed to operate not only on the microstates but also on their higher-level abstractions, and thus enabling the RL to flexibly change its perspective during the problem solving, each time choosing a perspective that leads it to the solution faster. In this paper, the algorithm for the Micro-Macro States combination is formulated and tested on path-finding problems in grid worlds. Here, the novelty of the proposed algorithm in hierarchically decomposing the given problems and in automatic goal-reaching in the sub-problem is tested against traditional RL and other hierarchical RL, and quantitatively analyzed.*

Keywords: Reinforcement learning, Hierarchical reinforcement learning, Task decomposition, Hierarchical abstraction

1. **Introduction.** In the past few years, Reinforcement Learnings (RL) [1] has been drawing much attention for solving problems that are difficult to be solved using other algorithms, for example, in [2-4]. The primary strength of RL is in its ability to directly interact with the learning environments without requiring explicit teacher signals that often are prohibitively difficult to obtain. However, traditional RL methods such as Temporal Difference (TD) [5], *Q-Learning* [6], and SARSA [7] sometimes require a prohibitively long time for learning complex tasks due to a high number of states and the rarity of rewards. Furthermore, the integration of RL and Deep Neural Network (DNN), called DRL [8-10], can achieve better performance compared to the traditional RL. Another way

DOI: 10.24507/ijicic.18.02.447

to improve the performance of RL is through Hierarchical Reinforcement Learning (HRL) that decomposes a complex problem into some more manageable smaller subtasks with their subgoals [11]. So far, there are mainly three ways for introducing hierarchy into RL.

Hierarchy in Actions: *Options* [12] and MAXQ [13,14] are approaches based on actions' hierarchy. *Options* and MAXQ were more optimal in accelerating agent learning to achieve their goals than traditional RL. These two approaches have distinct characteristics but commonly use state abstraction [15-19], temporal abstraction [12,20], and Semi Markov Decision Processes (SMDP) [21] for solutions to their subtasks. Further, studies on hierarchy in action were developed intensively with the objective of improving learning performance by automatically creating or discovering state abstractions [22-24], temporal abstractions [25-28], and subgoal on subtasks [23,26,29-35].

Hierarchy in Learning Agents: Several approaches that compose a hierarchy based on agents' control are Feudal Reinforcement Learning (FRL) [36], Feudal Networks (FuNs) [37], and Discrete EVent System Specification (DEVS) [38]. In these approaches, the upper layer agents give orders to the intermediate level agent. Subsequently, the intermediate-level agent gives orders to the lower-level agents that directly interact with the environment. Additionally, [36-38] have a characteristic in constructing macrostates by collecting adjacent microstates. Macrostates can be used to represent subtasks that are viewed as a new Markov Decision Processes (MDP) environment that is smaller than the actual environment [18,36].

Hierarchy in Environments: Hierarchical Abstract Machines (HAMs) [40] approach arranges a hierarchy based on the environment. In the HAMs, tasks or subtasks were called machines. Here, machines can interact with each other, in which the lower-level machines are tasks in the microstates, while machines at higher levels of abstraction represent subtasks. In HAMs, each machine will learn a policy to solve the subtasks and execute information exchange between them. Further research based on hierarchies in the environment was carried out by HAMQ-INT [41], *Q-Learning* for Reward Machines (QRM) [42], and HAM clustering [43].

The rest of this paper proposes a method of HRL by considering a hierarchy in environments. In comparison to the hierarchy in HAMs [40] where each hierarchy refers to a different environment, in this proposed method, the environment is hierarchically structured based on tasks in their original state (microstate) and subtasks represented by macrostates that are some collections of the microstate. A macrostate can be considered a higher-level abstraction of the microstates that allows RL to perceive the given problem from a different perspective. The ability to change perspective during the learning process enables HRL to choose problem representations that flexibly lead to a faster solution. Subtasks, represented by macrostates, also enable the proposed method to decompose problems into more manageable subproblems, allowing RL to solve difficult problems in a divide-and-conquer manner.

In this paper, the design of macrostates is similar to FRL [36] and DEVS [38], where the macrostate is a collection of adjacent microstates. However, in FRL [36] and DEVS [38], a microstate can only be part of one macrostate, while in the proposed method, a microstate can be a part of several different macrostates. The sharing of some microstates by several macrostates adds flexibility in forming the macrostates and enriches the diversity of perspectives that the RL can view. It is also intuitive to include a microstate with a high state-value into several macrostates to increase their state-value, thus improving the RL's likelihood to visit those macrostates during the learning process.

In the proposed HRL, agents need to learn and determine the best combination of microstates and macrostates to solve the given problem. Accordingly, the proposed algorithm is named Micro-Macro State Combination (MMSC) hierarchical reinforcement learning.

As in other HRL, in MMSC the learning agent first solves the subtask by reaching the subgoal. Here, the discovery of a subgoal in a subtask is essential. In *Options* [12], the subgoal is hand-designed, while in [22,23,26,29-34] the subgoals are automatically discovered. In those previous studies, in principle, the subgoal is designated by detecting the frequently visited states. In this study, the subgoal is automatically designed based on the maximum state-action value (Q value) of the microstates included in that macrostate at the current time. Another idea in the MMSC is to mix microstates and macrostates during the learning process to enrich the agent's choices and perspective.

The novelties of MMSC over the existing HRL methods are as follows: 1) allowing microstates to overlap over some macrostates resulting in increase flexibility; 2) discovery of subgoal automatically, thereby reducing additional hand-designed requirements; 3) mixing microstates and some macrostates as the domain for agent, enriching the agent's perspective in solving the problem.

This paper is organized as follows. Section 2 describes the MMSC algorithm in detail, while Section 3 is for explaining the experimental environment. Next, the results and evaluations of the experiment are discussed in Section 4. Finally, Section 5 describes the conclusions and future works for this study.

2. Micro-Macro States Combination Algorithm.

2.1. Framework.

Figure 1 illustrates the framework of the MMSC algorithm. The picture shows a grid environment with the size of 3×4 where the task of the agent is formulated into two layers. The lower layer depicts the task expressed by microstate, while

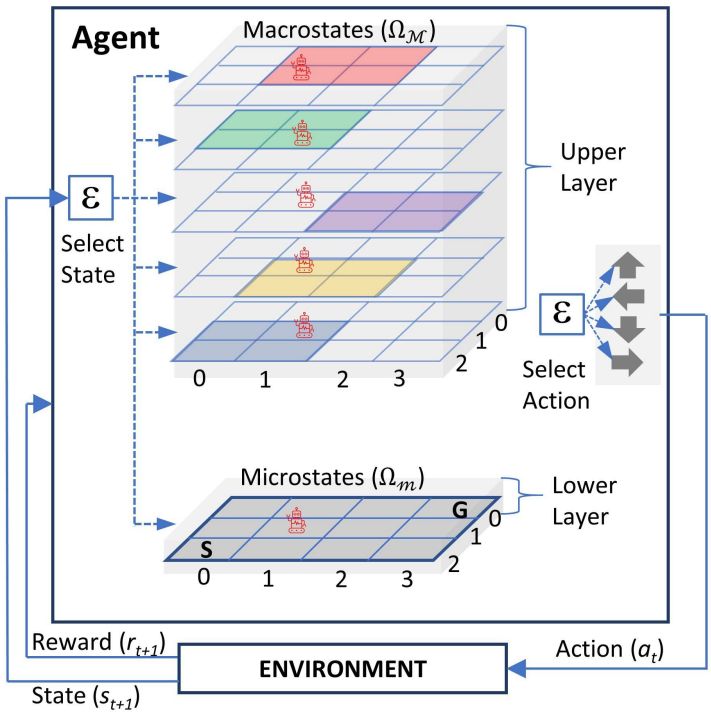

FIGURE 1. MMSC framework that illustrated the task decomposition into two hierarchies: the lower and upper layers. Here ϵ denotes ϵ-greedy policy.

the upper layer depicts the possible combinations of macrostates that represent subtasks of the original task. Here, a microstate, m, is characterized by (x, y), their coordinate in the problem space. The task of the agent in the picture is navigating from the start (denoted by S) at $(0, 2)$ to the goal (denoted by G) at $(3, 0)$ in the shortest path. Here, the set of all microstates is indicated by Ω_m, i.e., $m \in \Omega_m$. The size of the macrostates in this study is empirically determined to be 2×2, and hence each macrostate is a collection of four microstates. A macrostate is symbolized by $\mathcal{M}$, while the set of all macrostates is denoted by $\Omega_{\mathcal{M}}$, and hence $\mathcal{M} \in \Omega_{\mathcal{M}}$.

In the proposed MMSC, an agent can operate not only on the microstates but also on a higher-level abstraction of the microstates in the form of macrostates. Here, a macrostate constitutes a subtask. During the learning process, the agent learns the optimal combination of microstates and macrostates as a solution to solve the main task, and hence the problem space upon which it operates is $\Omega = \Omega_m \bigcup \Omega_{\mathcal{M}}$. The proposed MMSC is briefly presented in Algorithm 1. The MMSC is elaborated as follows, in which the number inside ( ) indicates the line number in Algorithm 1.

Algorithm 1. MMSC algorithm

1: Initialize $Q(s, a)$
2: **for** each episode **do**
3: Initialize s
4: **for** $t = 0$ to T **do**
5: Choose a conditioned on s using ϵ-greedy
6: Take action a, observe s', r
7: $m = s'$
8: $r_t = r$
9: **if** $s' \in \mathcal{M}_n^m$ **then**
10: Choose s' from $\{m, \mathcal{M}_n^m\}$ using ϵ-greedy
11: observe s'
12: **if** $s' = \mathcal{M}_n^m$ **then**
13: $I = m$
14: Define β based on Subsection 2.4
15: **for** $\tau = 0$ to $\mathcal{T}$ **do**
16: Policy on macrostate (Subsection 2.3)
17: **end for**
18: $G_{\mathcal{M}} = \sum_{k=0}^{\mathcal{T}} r_{\tau+k}$
19: $r_t = G_{\mathcal{M}}$
20: **end if**
21: **end if**
22: $Q^{new}(s, a) = Q^{old}(s, a) + \alpha \left(\mathcal{R}_s^a + \gamma \max_{a'} Q(s', a') - Q^{old}(s, a)\right)$
23: $s = s'$
24: **end for**
25: **end for**

Time Steps Diagram: Figure 2 illustrates the time diagram of MMSC. The agent interacts with the environment at each step at discrete time, t, by perceiving the state of the environment, $s_t \in \Omega$, which is tentatively represented by the microstate, m. For $t = 0$, s_t is the microstate, m, where the agent starts (line 3). While, for $t > 0$, the agent can change their perspective by selecting a macrostate, $\mathcal{M}$, for s_t.

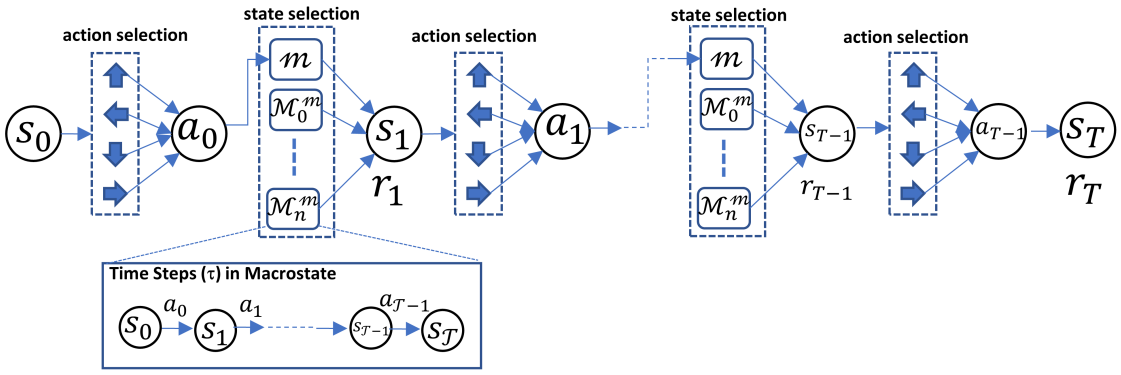

FIGURE 2. Time step t on MMSC and time steps τ on macrostate

After selecting state, s_t, the agent chooses an action symbolized by $a_t \in A$ where A is the set of all primitive actions. Action decision is based on the agent's objective of getting the maximum reward (lines 5-6).

Then, in response to the action chosen by the agent, the environment gives a reward, r_{t+1}, and a subsequent state, s_{t+1} (line 7). The next state, s_{t+1}, is temporarily represented by microstate, m, before the agent performs the state selection described previously (line 8).

Based on the sequence of processes, the probability of moving to the next state, s_{t+1}, depends only on the current state, s_t, and does not depend on the previous state. Therefore, the state transition probability equation is expressed as follows:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a] \tag{1}$$

Based on the macrostate formation in Subsection 2.2, each microstate can be part of a maximum of four macrostates of size 2×2. Hence, at time t, the agent can learn to select a maximum of five different perspectives (1 microstate and 4 macrostates) as its state. The macrostates that can be formed from a microstate, m, are denoted by $\mathcal{M}^m_n$, in which $0 \leq n \leq 3$ denotes the index of that macrostates (line 9). The probability that $s \in \{m, \mathcal{M}^m_n\}$ is adopted as the state when the agent is at m at time t is (line 10)

$$\mathcal{P}_s = \mathbb{P}[s_t = s | m] \tag{2}$$

The selection of the state at time t is based on the agent's objective to obtain the maximum reward:

$$\mathcal{R}_s = E[r_{t+1} | s_t = s] \tag{3}$$

Here, r_{t+1} denotes the reward at time $t+1$, while $E[r_{t+1}|s_t = s]$ denotes the expectation. Here, ϵ-greedy is used as a policy to select a state.

Total Reward, Policy, and Value Function: Transferring the state from s_t to s_{t+1} allows the agent to receive the reward from an environment as follows:

$$\mathcal{R}^a_{ss'} = E[r_{t+1} | s_{t+1} = s', s_t = s, a_t = a] \tag{4}$$

The total reward, G_t, received by the agent from the environment is as follows:

$$G_t = \sum_{k=1}^{T} r_{t+k} \tag{5}$$

Here, T is the terminal time step. The purpose of MMSC is to maximize the total amount of rewards received by an agent from the initial state to the terminal state. Calculation of

the total amount of rewards on MMSC requires an additional concept of discount factor, $0 \leq \gamma \leq 1$. So with the discount factor, (5) becomes

$$G_t = \sum_{k=1}^{T} \gamma^k r_{t+k} \tag{6}$$

Here, the value of the reward, r, at each step depends on the next state, s_{t+1}, perceived by the agent. The designer only defines the reward value for the next state, the provisional state, which is a microstate. Suppose a macrostate subsequently replaces the provision microstate for s_{t+1} (line 12). In that case, the value of the reward is based on the number of steps made by the agent from the initial state, I, to the terminal condition, β, in the macrostate. I and β are described in Subsection 2.4 for autonomous subgoal discovery (lines 13-14). In addition to the diagram of the overall agent learning process time, Figure 2 also shows step-by-step procedures for the agent to complete the subtask. Completion of the subtask depends on the agent's policy to choose the action, a_τ, which is explained in Subsection 2.3 (line 16). Therefore, the agent does not learn when solving the subtask, and the calculation of the reward value does not require a discount factor. Hence, if the next state, s_{t+1}, perceived by the agent is a macrostate, the reward value, $G_{\mathcal{M}}$, can be calculated as follows (line 18):

$$G_{\mathcal{M}} = \sum_{k=0}^{\mathcal{T}} r_{\tau+k} \tag{7}$$

The value function estimates the expected reward when the agent is currently in state s_t:

$$V_\pi(s) = E_\pi[G_t | s_t = s] \tag{8}$$

Equation (8) is also known as the state value function, $V_\pi(s)$, which can be divided into two parts, namely the immediate reward and the discounted value of the successor state that is written as follows:

$$\begin{aligned} V_\pi(s) &= E_\pi\left[G_t | s_t = s\right] \\ &= E_\pi\left[\sum_{k=1}^{T} \gamma^k r_{t+k} | s_t = s\right] \\ &= E_\pi[r_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= E_\pi[r_{t+1} | s_t = s] + E_\pi[\gamma V_\pi(s_{t+1}) | s_t = s] \end{aligned} \tag{9}$$

In addition to the state value function, there is a state-action value function, $Q_\pi(s, a)$, that is expressed as follows:

$$\begin{aligned} Q_\pi(s, a) &= E_\pi[G_t | s_t = s, a_t = a] \\ &= E_\pi[r_{t+1} | s_t = s, a_t = a] + E_\pi[\gamma V_\pi(s_{t+1}) | s_t = s, a_t = a] \end{aligned} \tag{10}$$

Further the relation between $V_\pi(s)$ and $Q_\pi(s, a)$ can be written as follows:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s, a) \tag{11}$$

From (4) and (10), $Q_\pi(s, a)$ becomes

$$Q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V_\pi(s') \tag{12}$$

$\mathcal{P}^a_{ss'}V_\pi(s')$ defines a state value function of $s' = s_{t+1}$ which depends on the state transition from $s = s_t$ to $s' = s_{t+1}$ based on the selected action a at time t. The Bellman equation for $V_\pi(s)$ is

$$V_\pi(s) = \sum_{a\in A} \pi(a|s) \left(\mathcal{R}^a_s + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} V_\pi(s') \right) \tag{13}$$

The Bellman equation calculates the value of the current state depends only on the possible next states. Similarly, $Q_\pi(s,a)$ can also be written using the Bellman equation [1,21,44].

The best performance of MDP is determined by the optimal value function and optimal policy, in which the optimal value function is

$$V^* = \max_\pi V_\pi(s) \tag{14}$$

while the optimal policy is

$$Q^* = \max_\pi Q_\pi(s,a) \tag{15}$$

Temporal Difference: The temporal difference (TD), is an agent learning approach for successively updating the state-action values, $Q(s,a)$. The state-action value update is based on the reward received and the next expected reward by the agent calculated as follows:

$$\begin{aligned} TD(a,s) &= Q^{new}(s,a) - Q^{old}(s,a) \\ &= \mathcal{R}^a_s + \gamma \sum_{s'\in S} \mathcal{P}^a_{ss'} Q^*(s',a') - Q^{old}(s,a) \\ &= \mathcal{R}^a_s + \gamma \max_{a'} Q(s',a') - Q^{old}(s,a) \end{aligned} \tag{16}$$

Here, a' is defined as the action for which the maximum reward is attained. Equation (16) provides a temporal difference in the value of Q, which can help capture changes that occur in the environment. Then, the updated state-action value, $Q^{new}(s,a)$ is written as (line 22)

$$\begin{aligned} Q^{new}(s,a) &= Q^{old}(s,a) + \alpha TD(a,s) \\ &= Q^{old}(s,a) + \alpha \left(\mathcal{R}^a_s + \gamma \max_{a'} Q(s',a') - Q^{old}(s,a) \right) \end{aligned} \tag{17}$$

Here, $0 \le \alpha \le 1$ is the empirically set learning rate.

2.2. **The formation and components of macrostate.**

Macrostate Formation: The combination of macrostates in which a microstate is a member is denoted by $\mathcal{M}^m_n$. In Figure 1, where the agent is located at $m = (1,1)$, the macrostates that can be formed are $\mathcal{M}^m_0 = \{(1,0),(1,1),(2,0),(2,1)\}$, $\mathcal{M}^m_1 = \{(1,1),(1,2),(2,1),(2,2)\}$, $\mathcal{M}^m_2 = \{(0,0),(0,1),(1,0),(1,1)\}$, and $\mathcal{M}^m_3 = \{(0,1),(0,2),(1,1),(1,2)\}$.

Macrostate Components: A macrostate has three components: an initial state, I, a terminal condition, β, and policies, $\pi_{\mathcal{M}}$. Here, the initial state $s_0 = I = m$. The policy on state selection to change agent perspective has been described in Subsection 2.1. After the agent adopts one of the macrostates as its current state, the agent takes action based on the policy, $\pi_{\mathcal{M}}$, until the agent reaches the terminal condition, β, that is the subgoal of that macrostate. Here, $\pi_{\mathcal{M}}$ policies are deterministic and hand-coded, while the subgoal is automatically set.

2.3. **Policy on macrostate ($\pi_{\mathcal{M}}$).** A microstate that represents an obstacle cannot be included in a macrostate, and hence all the macrostates are obstacle-free. The obstacle-free characteristic allows the agent to navigate around obstacles carefully and thus improve the expected rewards. Algorithm 2 is a policy designed on macrostates in a 2-dimensional environment to determine the actions selected by the agent. In the macrostate, the agent task starts from the initial state, I, and ends at the terminal condition, β.

Algorithm 2. Policy on macrostate

```
x = I[0] − β[0]
y = I[1] − β[1]
if y < 0 then
    action ← down
else if y > 0 then
    action ← up
else if x < 0 then
    action ← right
else if x > 0 then
    action ← left
end if
```

2.4. **Autonomous subgoal discovery.** Subsection 2.1 explains that the state-action value function determines how well the agent performs the action in the state, s_t, with the policy, π. At each step t, the state-action value function, the Q function in the state, s_t, is updated using Equation (17). Because the state, s_t, which is a subset of the whole set of microstates and macrostates, the Q value of the chosen microstate or macrostate will be updated during the learning process.

As an illustration, Figure 3 shows a problem in a 3×4 grid world where the agent task is formulated in two layers. The lower layer depicts a snapshot of the Q-values of the microstates, while the upper layer depicts all the macrostates that can be formed from these microstates.

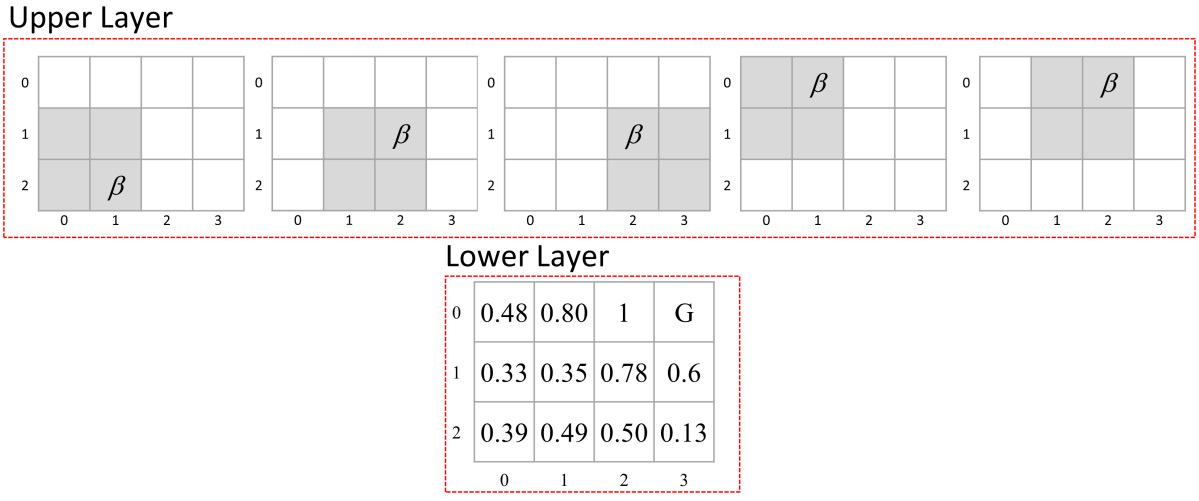

FIGURE 3. Autonomous subgoal discovery. Label G indicates goal in the agent's task, while label β are terminal conditions.

Furthermore, the subgoal in the subtask, represented by the terminal condition β, is determined based on the maximum Q value of the microstates that constitute a particular macrostate. For example, in Figure 3, one of the macrostates $\mathcal{M}_0^m = \{(1,0),(1,1),(2,0),(2,1)\}$ is the collection of microstates $(1,0)$, $(1,1)$, $(2,0)$, and $(2,1)$ that each of microstates has Q value of 0.80, 0.35, 1, and 0.78. The Q value in microstate $(2,0)$ has the largest value, so microstate $(2,0)$ is selected as the terminal condition, β, which represents the subgoal in the subtask.

3. **Experiments.** This study was evaluated in the grid-world shown in Figure 4. Figures 4(a)-4(d) represent environments consisting of four rooms where all rooms include two hallways but have different complexity in the obstacles distributions. The proposed MMSC is tested against *Q-Learning* [9] and other HRL, *Options* [12].

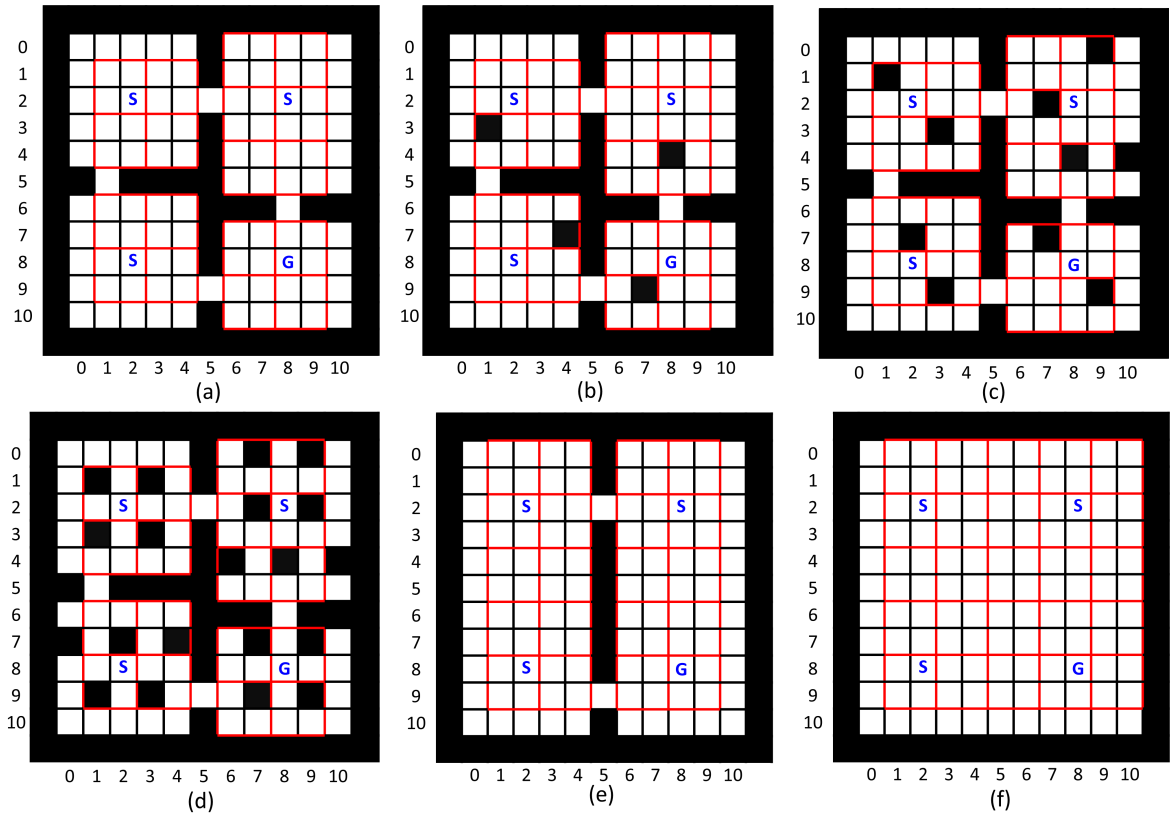

FIGURE 4. (color online) The grid-world environments. The red border indicates the subenvironments in *Options*.

In the original *Options*, the environment is divided into a number of subenvironments, for example in Figure 4, the numbers of rooms. For assuring comparison's fairness, here *Options* are calibrated to have many more hand-designed subenvironments.

In these experiments, the task of the agent is to find the shortest path from the initial state labeled S to the goal labeled G. During the learning process, the agent will return to the initial state if the agent reaches the goal or hits a wall. Each environment has been tested on different initial states: $(2,2)$, $(2,8)$, and $(8,2)$, but with the same goal at $(8,8)$.

Here, the four primitive actions of the agent are up, down, left, and right. In the experiments, the discount factor (γ), learning rate (α), epsilon (ϵ), and rewards are empirically determined. In the experiment, $\gamma = 0.9$ to ensure that future rewards are more important than the past, $\alpha = 0.1$ to ensure gradual exploitation of the environment. While the parameters are empirically chosen, they are fixed for various problems. The fixed parameter

values regarding different problems indicate that the proposed method is not excessively sensitive to the parameters' choice, thus ensuring generality.

Furthermore, this experiment used $\epsilon = 0.999^{episodes}$. This formula is intended to do more exploration at the beginning of the episode and exploitation at the end. Moreover, the reward of -0.1 is given when the agent moves to a new state. While -1 when the agent hits an obstacle, and $+1$ when the agent reaches the goal.

4. **Results and Evaluations.** The efficiency of the proposed MMSC is evaluated regarding the number of steps until the goal is reached or an obstacle is hit, as well as the success rate. Here, MMSC is compared against *Q-Learning* and the modified *Options* on the learning environments shown in Figure 4. Figure 5 shows some of the experiment results on the environments in Figure 4(a), Figure 4(d), and Figure 4(f), from two different

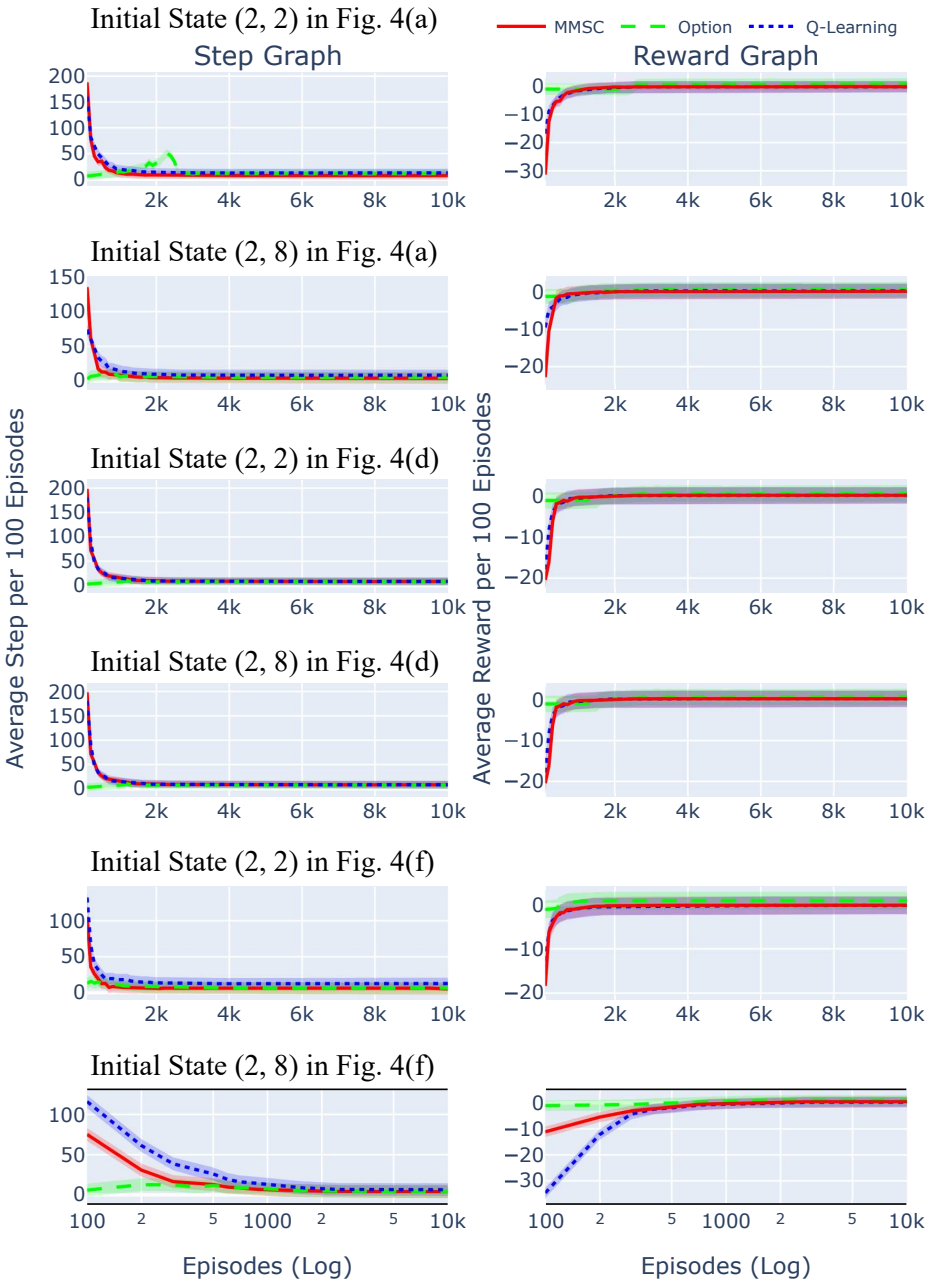

FIGURE 5. The steps and rewards graph of MMSC, *Options*, and *Q-Learning*

initial conditions. On the step graph, the average number of steps per 100 episodes is displayed on the vertical axis while the episodes are on the horizontal axis. In the reward graphs, the vertical axis is the average reward per 100 episodes, while the horizontal axis represents the episodes. The number of steps in RL shows the number of decisions made by the agent from the initial state to the goal. Here, the efficacy of learning algorithms is indicated by a fewer number of steps.

Table 1 shows the average number of steps generated in episodes 9,000 to 10,000 during the learning process in MMSC, *Q-Learning*, and *Options* across all problems. The number of steps was averaged in the 9,000 to 10,000 episodes because the learning in those episodes was stable. The success rate of the agent during the learning process is also shown in Table 1. Table 2 shows the efficiency of the agent to reach the goal based on the complexity of the obstacles distributions and the distance from the initial state to the goal. Using *Q-Learning* as the standard, the efficiency of MMSC and *Options* is calculated as follows:

$$\eta = \left(1 - \frac{T}{T_{QL}}\right) \times 100\% \tag{18}$$

Here, T is the total number of steps of MMSC or *Options* for which efficiency is to be calculated, and T_{QL} is the total number of steps of *Q-Learning*.

TABLE 1. The average number of steps and success rate in episodes 9,000 until 10,000

Environments	**Initial state**	**Algorithms**					
		MMSC		***Q-Learning***		***Options***	
		Number of steps	**Success rate**	**Number of steps**	**Success rate**	**Number of steps**	**Success rate**
Figure 4(a)	(2, 2)	**7.44**	**89.99%**	12.67	81.23%	11.25	67.97%
	(2, 8)	**4.31**	**97.45%**	8.53	92.00%	5.29	86.30%
	(8, 2)	**5.16**	**90.91%**	6.36	90.84%	5.35	82.40%
Figure 4(b)	(2, 2)	**8.50**	95.20%	12.78	**98.36%**	10.22	61.49%
	(2, 8)	**6.51**	**99.60%**	8.49	99.59%	7.23	76.92%
	(8, 2)	**6.45**	98.21%	8.49	**98.66%**	8.10	74.08%
Figure 4(c)	(2, 2)	**9.57**	94.86%	12.62	**95.10%**	10.60	52.63%
	(2, 8)	7.37	97.22%	8.46	**97.54%**	**7.06**	71.21%
	(8, 2)	**7.40**	**96.86%**	8.43	96.66%	7.96	73.76%
Figure 4(d)	(2, 2)	12.75	**98.10%**	12.75	**98.10%**	**12.54**	45.49%
	(2, 8)	8.46	**99.64%**	8.52	99.17%	**8.20**	67.29%
	(8, 2)	8.37	99.30%	8.41	**99.50%**	**8.10**	72.6%
Figure 4(e)	(2, 2)	**7.29**	**96.60%**	12.69	96.37%	9.63	77.79%
	(2, 8)	**5.21**	98.77%	8.49	**99.20%**	5.29	87.09%
	(8, 2)	4.29	98.30%	6.48	**98.68%**	**3.23**	95.09%
Figure 4(f)	(2, 2)	**5.78**	91.22%	12.69	92.60%	6.49	**94.92%**
	(2, 8)	**4.12**	95.74%	6.47	**98.25%**	4.24	95.21%
	(8, 2)	4.09	95.75%	6.50	**98.27%**	**3.23**	96.02%

From the step graph, it can be observed that all the algorithms succeeded in minimizing the number of steps, thus maximizing the efficiency in reaching the goal along with the progress of their learning. It can also be observed that MMSC and *Q-Learning* need more steps than *Options* to explore the environments in the early episodes. The fewer steps of *Options* in the early episodes do not necessarily mean that the agent successfully reaches the goal but also includes conditions when it hits obstacles in the environment. From

TABLE 2. The average of efficiency

Category	Environments	Efficiency	
		MMSC	*Options*
Complexity of the obstacles distributions	Figure 4(a)	**38.64%**	20.57%
	Figure 4(b)	**27.89%**	14.15%
	Figure 4(c)	**17.52%**	13.18%
	Figure 4(d)	0.34%	**2.83%**
	Figure 4(e)	**39.30%**	34.38%
	Figure 4(f)	**71.43%**	45.60%
Path length from initial state to the goal	$(2, 2)$	**32.64%**	20.30%
	$(2, 8)$	**26.51%**	23.79%
	$(8, 2)$	**19.95%**	19.48%

Table 1, the average success rates of MMSC, *Q-Learning*, and *Options* in achieving the goal is 96%, 96%, and 77%, which means that *Options* most often hit obstacles during the learning process.

In Figure 4(a) and Figure 4(f), MMSC and *Options* as HRL methods have fewer steps than *Q-Learning* in the later episodes meaning that HRL performed better than *Q-Learning*. Using *Q-Learning* as a standard, the average, increase in efficiency of MMSC and *Options* are 28% and 21%, respectively as indicated in Table 2. However, in Figure 4(d), MMSC and *Options* have almost the same number of steps as *Q-Learning*. The results indicate that MMSC and *Options* as HRL methods have the same performance as *Q-Learning* if the environment has more obstacles. It can also be observed in Table 2 that MMSC and *Options* have an average improved efficiency of less than 3% in Figure 4(d), while in the other environments it is above 13%. Many obstacles in the environment cause MMSC and *Options* to fail to set a macrostate or "option" resulting in a similar performance to *Q-Learning*.

In addition, the difference between MMSC and *Options* can be identified clearly at the initial state $(2, 2)$ in Figure 4(a) and Figure 4(f). MMSC seems to have fewer steps than *Options*, so MMSC performs better than *Options* in the environment. Table 2 shows that the efficiency of MMSC against *Q-Learning* increases when the initial state is further away from the goal. Because in MMSC, the agent perceives more macrostates to be learned than the number of "option" in *Options*. However, in the initial state $(2, 8)$, the numbers of steps owned by MMSC and *Options* are almost the same because the distance between the initial state and the goal is closer, causing the macrostates learned by the agent to decrease.

The purpose of the agent learning through RL is to obtain the maximum reward. It can be observed from Figure 5 that the agent rapidly reached the maximum reward. In Figure 4(a) and Figure 4(d), *Options* shows that the agent learns longer than MMSC and *Q-Learning* to reach the goal. However, in Figure 4(f), MMSC, *Q-Learning*, and *Options* have almost the same learning time to reach the goal. Table 1 shows that the presence of obstacles will also significantly reduce the success rate of *Options* compared to MMSC and *Q-Learning*.

Figure 6 shows histograms of Q value to illustrate the optimal policies of the agent to achieve the goal. The Q value for each microstate is calculated based on the average Q value between the microstates and the macrostates associated with those microstates. An optimal policy can be visualized by following the lighter-colored state from the start to the goal.

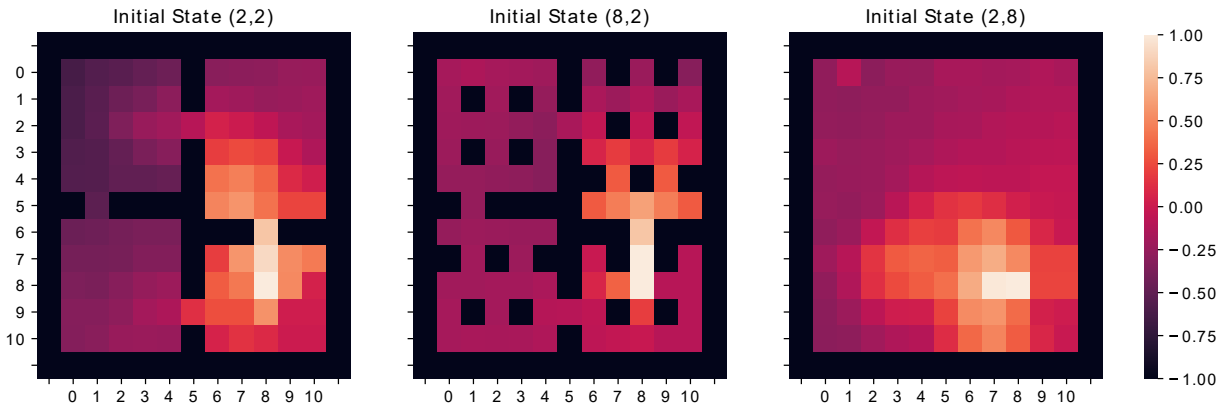

FIGURE 6. (color online) Histogram of Q values show the paths resulting from agent learning in three different environments. Initial states $(2, 2)$, $(8, 2)$, and $(2, 8)$ use the environment in Figures 4(a), 4(d), and 4(f) respectively.

5. **Conclusions.** In this paper, the proposed MMSC algorithm offers a new mechanism for HRL in which the task is formulated into two layers of abstraction. The lower layer depicts tasks in their original state represented by microstates, while the upper layer depicts macrostates as representations of subtasks. In this paper, a macrostate is hand-designed with a size of 2×2 and is a collection of four neighboring microstates. Here, the subgoal is represented by the terminal condition in the macrostate. Subgoal discovery is achieved automatically based on the maximum Q value among the microstates that make up the macrostate. The efficiency of the number of steps and the agent's success rate in achieving goals during the learning process is influenced by the number of obstacles in the environment and the distance from the initial state to the goal. In MMSC, the combination of microstates and macrostates increases the agent's perspective in learning the environment.

In MMSC, the addition of macrostates into the state selection does not aggravate the curse of dimensionality. The macrostates are constrained around the microstate currently visited by the learning agent and hence do not expand the dimensionality of the problem in an unconstrained manner. The addition of the macrostates offers new choices for the agent to exploit some constrained areas, in which some high-value states attract more attention. Furthermore, the hand-designed policy in macrostate allows the inclusion of heuristics and humans apriori knowledge into RL, contributing to increasing the learning efficiency as apparent from the experimental results.

For future research, several aspects need to be investigated: 1) adaptive size of macrostates based on the complexity of the environment; 2) generation of the deeper hierarchy of abstraction; and 3) implementations to real-world problems, including robotics applications.

Acknowledgment. This work was supported by JSPS Grants-in-Aid for Scientific Research on Innovative Areas (Research in a proposed research area) 18H05473 and 18H05895.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, The MIT Press Cambridge, Massachusetts, London, England, 1998.

[2] Z. Li, J. Liu, Z. Huang, Y. Peng, H. Pu and L. Ding, Adaptive impedance control of human-robot cooperation using reinforcement learning, *IEEE Trans. Industrial Electronics*, vol.64, no.10, pp.8013-8022, DOI: 10.1109/TIE.2017.2694391, 2017.
[3] D. L. Leottau, J. Ruiz-del-Solar and R. Babuška, Decentralized reinforcement learning of robot behaviors, *Artificial Intelligence*, vol.256, pp.130-159, 2018.
[4] L. Guo, S. A. A. Rizvi and Z. Lin, Optimal control of a two-wheeled self-balancing robot by reinforcement learning, *International Journal Robust Nonlinear Control*, vol.31, pp.1885-1904, 2021.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, Human-level control through deep reinforcement learning, *Nature*, vol.518, pp.529-533, 2015.
[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, Mastering the game of Go with deep neural networks and tree search, *Nature*, vol.529, pp.484-489, 2016.
[7] R. Watanuki, T. Horiuchi and T. Aodai, Vision-based behavior acquisition by deep reinforcement learning in multi-robot environment, *ICIC Express Letters, Part B: Applications*, vol.11, no.3, pp.237-244, 2020.
[8] R. S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning*, vol.3, no.1, pp.9-44, 1988.
[9] C. J. C. H. Watkins and P. Dayan, Q-Learning, *Machine Learning*, vol.8, pp.279-292, 1992.
[10] G. A. Rummery and M. Niranjan, *On-Line Q-Learning Using Connectionist Systems*, Cambridge University Engineering Department, 1994.
[11] Y. Lu, X. Xu, X. Zhang, L. Qian and X. Zhou, Hierarchical reinforcement learning for autonomous decision making and motion planning of intelligent vehicles, *IEEE Access*, vol.8, pp.209776-209789, 2020.
[12] R. S. Sutton, D. Precup and S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, *Artificial Intelligence*, vol.112, no.1, pp.181-211, 1998.
[13] T. G. Dietterich, The MAXQ method for hierarchical reinforcement learning, *The 15th International Conference on Machine Learning*, pp.118-126, 1998.
[14] T. G. Dietterich, An overview of MAXQ hierarchical reinforcement learning, in *Abstraction, Reformulation, and Approximation. SARA 2000. Lecture Notes in Computer Science*, B. Y. Choueiry and T. Walsh (eds.), Berlin, Heidelberg, Springer, 2000.
[15] L. Li, T. J. Walsh and M. L. Littman, Towards a unified theory of state abstraction for MDPs, *The 9th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, pp.531-539, 2006.
[16] M. Ponsen, M. E. Taylor and K. Tuyls, Abstraction and generalization in reinforcement learning: A summary and framework, in *Adaptive and Learning Agents. ALA 2009. Lecture Notes in Computer Science*, M. E. Taylor and K. Tuyls (eds.), Berlin, Heidelberg, Springer, 2010.
[17] J. Menashe and P. Stone, State abstraction synthesis for discrete models of continuous domains, *Data Efficient Reinforcement Learning Workshop at AAAI Spring Symposium*, Stanford, CA, USA, pp.331-337, 2018.
[18] D. Abel, D. Arumugam, L. Lehnert and M. L. Littman, State abstractions for lifelong reinforcement learning, *Proc. of the 35th International Conference on Machine Learning*, pp.10-19, 2018.
[19] D. Misra, M. Henaff, A. Krishnamurthy and J. Langford, Kinematic state abstraction and provably efficient rich-observation reinforcement learning, *Proc. of the 37th International Conference on Machine Learning (ICML)*, pp.6917-6927, 2020.
[20] D. Precup, R. S. Sutton and S. Satinder, Theoretical results on reinforcement learning with temporally abstract options, in *Machine Learning: ECML-98. ECML 1998. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, C. Nédellec and C. Rouveirol (eds.), Berlin, Heidelberg, Springer, 1998.
[21] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New Jersey, 1994.
[22] T. G. Dietterich, Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research*, vol.13, no.3, pp.227-303, 2000.
[23] D. Jardim, L. Nunes and S. Oliveira, Hierarchical reinforcement learning: Learning sub-goals and state-abstraction, *Proc. of the 6th Iberian Conference on Information Systems and Technologies (CISTI2011)*, pp.1-4, 2011.

[24] C. Allen, N. Parikh, O. Gottesman and G. Konidaris, Learning Markov state abstractions for deep reinforcement learning, *NeurIPS Workshop on Deep Reinforcement Learning*, 2020.
[25] D. Precup and R. S. Sutton, Multi-time models for temporally abstract planning, *Advances in Neural Information Processing Systems*, pp.1050-1056, 1998.
[26] Ö. Şimşek and A. G. Barto, Using relative novelty to identify useful temporal abstractions in reinforcement learning, *ACM International Conference Proceeding Series*, vol.69, pp.751-758, 2004.
[27] P. L. Bacon, J. Harb and D. Precup, The option-critic architecture, *Proc. of the 31st AAAI Conference on Artificial Intelligence*, pp.1726-1734, 2017.
[28] Biedenkapp, R. Rajan, F. Hutter and M. Lindauer, TempoRL: Learning when to act, *Proc. of the 38th International Conference on Machine Learning*, pp.914-924, 2021.
[29] B. L. Digney, Learning hierarchical control structures for multiple tasks and changing environments, *Proc. of the 5th International Conference on Simulation of Adaptive Behavior on from Animals to Animats*, pp.321-330, 1998.
[30] E. A. Mcgovern and A. G. Barto, Automatic discovery of subgoals in reinforcement learning using diverse density, *Proc. of the 18th International Conference on Machine Learning*, 2001.
[31] B. Hengst, Discovering hierarchy in reinforcement learning with HEXQ, *Proc. of the 19th International Conference on Machine Learning (ICML)*, pp.243-250, 2002.
[32] Menache, S. Mannor and N. Shimkin, Q-Cut – Dynamic discovery of sub-goals in reinforcement learning, in *Machine Learning: ECML 2002. ECML 2002. Lecture Notes in Computer Science*, T. Elomaa, H. Mannila and H. Toivonen (eds.), Berlin, Heidelberg, Springer, 2002.
[33] S. Mannor, I. Menache, A. Hoze and U. Klein, Dynamic abstraction in reinforcement learning via clustering, *Proc. of the 21st International Conference on Machine Learning (ICML)*, New York, USA, 2004.
[34] Ö. Şimşek, A. P. Wolfe and A. G. Barto, Identifying useful subgoals in reinforcement learning by local graph partitioning, *ACM International Conference Proceeding Series*, vol.119, pp.817-824, 2005.
[35] D. Xiao, Y. T. Li and C. Shi, Autonomic discovery of subgoals in hierarchical reinforcement learning, *Journal of China Universities of Posts and Telecommunications*, vol.21, no.5, pp.94-104, 2014.
[36] P. Dayan and G. Hinton, Feudal reinforcement learning, *Advances in Neural Information Processing Systems*, pp.271-278, 1993.
[37] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver and K. Kavukcuoglu, FeUdal networks for hierarchical reinforcement learning, *Proc. of the 34th International Conference on Machine Learning (ICML)*, vol.7, pp.5409-5418, 2017.
[38] C. Kesler, L. Capocchi and J.-F. Santucci, Hierarchical Markov decision process based on DEVS formalism, *Proc. of the 2017 Winter Simulation Conference*, pp.1001-1012, 2017.
[39] R. E. Bellman, Markovian decision processes, *Journal of Mathematics and Mechanics*, vol.6, no.5, pp.679-684, 1957.
[40] R. Parr and S. Russell, Reinforcement learning with hierarchies of machines, *Advances in Neural Information Processing Systems*, Cambridge, MA, pp.1043-1049, 1998.
[41] A. Bai and S. Russell, Efficient reinforcement learning with hierarchies of machines by leveraging internal transitions, *IJCAI International Joint Conference on Artificial Intelligence*, pp.1418-1424, 2017.
[42] R. T. Icarte, T. Q. Klassen, R. Valenzano and S. A. McIlraith, Using reward machines for high-level task specification and decomposition in reinforcement learning, *Proc. of the 35th International Conference on Machine Learning (ICML)*, pp.3347-3358, 2018.
[43] S. Alexey and A. I. Panov, Hierarchical reinforcement learning with clustering abstract machines, *RCAI: Russian Conference on Artificial Intelligence*, pp.30-43, 2019.
[44] R. E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

Author Biography

Gembong Edhi Setyawan received the B.Eng. degree from the Department of Electrical Engineering of Brawijaya University and the M.Eng. degree from the Department of Electrical Engineering of Sepuluh Nopember Institute of Technology in Indonesia. He is now pursuing a Ph.D. at the Department of Pure and Applied Physics of Waseda University in Tokyo, Japan. Since 2012, he has been a lecturer at the Department of Computer Engineering of Brawijaya University in Indonesia. His primary research interests are reinforcement learning theory and applications, as well as intelligent robotic control.

Hideyuki Sawada is a Professor in the Department of Applied Physics, Faculty of Science and Engineering, Waseda University. He received the B.E., M.E. and Ph.D. degrees in applied physics from Waseda University in 1990, 1992 and 1999, respectively. In 1999, he joined the Department of Intelligent Mechanical Systems Engineering, Faculty of Engineering, Kagawa University, as an Associate Professor, and became a Professor in 2010. His current research interests include robotics, sound and image processing, machine learning, human interfaces, and tactile sensing and display.

Pitoyo Hartono received the B.Eng., M.Eng., and D.Eng. degrees from the Department of Pure and Applied Physics, Waseda University, Tokyo, Japan, in 1993, 1995, and 2002, respectively. He was a Software Engineer with Hitachi Ltd., from 1995 to 1998. From 2001 to 2005, he was a Research Associate and a Visiting Lecturer with Waseda University. He was an Associate Professor with Future University Hakodate, Hakodate City, Japan, from 2005 to 2010. Since 2010, he has been a Professor with the School of Engineering, Chukyo University, Nagoya, Japan. His research interests include the theory and application of neural networks, explainable AI, intelligent robotics, and man-machine interface.