PAPER Special Section on Circuits and Systems

Reinforcement Learning for Multi-Agent Systems with Temporal Logic Specifications

Keita TERASHIMA[†], *Nonmember*, **Koichi KOBAYASHI**[†a)], *and* **Yuh YAMASHITA**[†], *Members*

SUMMARY In a multi-agent system, it is important to consider a design method of cooperative actions in order to achieve a common goal. In this paper, we propose two novel multi-agent reinforcement learning methods, where the control specification is described by linear temporal logic formulas, which represent a common goal. First, we propose a simple solution method, which is directly extended from the single-agent case. In this method, there are some technical issues caused by the increase in the number of agents. Next, to overcome these technical issues, we propose a new method in which an aggregator is introduced. Finally, these two methods are compared by numerical simulations, with a surveillance problem as an example.

key words: *multi-agent systems, reinforcement learning, linear temporal logic, aggregator, surveillance*

1. Introduction

A multi-agent system (MAS) [1] is a system composed of multiple autonomous agents that interact with each other and with their environment. Within a group of agents in an MAS, each agent has an individual or common goal. If there is a common goal among agents, it is a challenge how they cooperate with each other in order to achieve the goal. Reinforcement learning (RL) [2] is well known as one of the learning methods used in the design of an MAS. RL targets an unknown environment with uncertainty and aims to acquire optimal control strategies by learning through repeated actions by agents in the environment. RL is a very effective method for designing large-scale and complex systems, because knowledge of the environment is not required for a designer of the system. In multi-agent reinforcement learning (MARL), agents observe other agents as part of their environment. Therefore, as the number of agents increases, the state space grows exponentially, and the explosion of the state space causes a significant decrease in learning speed.

On the other hand, one of the recent trends is to describe control specifications by linear temporal logic (LTL) [3], and to design a controller by RL under the temporal logic constraints. Temporal logic is a logical system that is added temporal operators to standard logic operations. Using temporal logic formulas, we can describe properties related to time. Various methods have been proposed for MARL using temporal logic, regardless of the type of temporal logic [4]–[7].

In this paper, we propose two novel MARL methods, where the control specification is described by an LTL formula. First, we propose a simple solution method, which is extended directly from the single-agent case [8]. In this method, there are some technical issues such as the explosion of the state space caused by the increase in the number of agents. Then, we propose a distributed method with an aggregator to solve this problem. In MARL, the Centralized Training with Decentralized Execution (CTDE) methods are known [9], [10]. In the CTDE method, a kind of aggregators is introduced during training. However, to the best of our knowledge, temporal logic has not been considered in the existing methods. Finally, these two learning methods are compared by numerical simulations, with a surveillance problem as an example. It is shown that there are differences in the state space and learning speed, and the performance of the latter method is better than that of the former method.

This paper is organized as follows. In Sect. 2, we describe a Markov Decision Process, LTL, and automata. In Sect. 3, we propose two novel MARL methods with LTL specifications. In Sect. 4, a numerical example is presented to show our proposed methods. In Sect. 5, we conclude this paper.

Notation: Let $\mathbb{N}_0$ denote the set of nonnegative integers. Let $\mathbb{R}_{\geq 0}$ denote the set of nonnegative real numbers. Let ε denote the empty string.

2. Preliminaries

In this section, first, as preliminaries for MARL, we explain the outline of a Markov Decision Process (MDP) for the single-agent case. Next, we describe how to describe the control specifications for the agents using LTL and how to convert an LTL formula into a limit-deterministic generalized Büchi automaton (LDGBA).

2.1 Markov Decision Process

In this paper, interactions between an agent and an environment are represented by an MDP defined below.

Definition 1. *For the single-agent case, a (labeled) MDP is a tuple* $M = (S, A, s_{init}, P, AP, L)$*, where* S *is a finite set of states,* A *is a finite set of actions,* $s_{init} \in S$ *is the initial state,* $P : S \times A \times S \to [0, 1]$ *is a transition probability function,* AP

Manuscript received February 3, 2023.
Manuscript revised May 24, 2023.
Manuscript publicized July 19, 2023.

[†]The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060-0814 Japan.
a) E-mail: k-kobaya@ssi.ist.hokudai.ac.jp

DOI: 10.1587/transfun.2023KEP0016

is a finite set of atomic propositions, and $L : S \times A \times S \to 2^{AP}$ *is a labeling function labeling each transition with a subset of atomic propositions. Let* $\mathcal{A}(s) = \{a \in A; \exists s' \in S \text{ s.t. } P(s'|s,a) \neq 0\}$*, and* $\Sigma_{s' \in S} P(s'|s,a) = 1$ *for any* $s \in S$ *and* $a \in \mathcal{A}(s)$*.*

2.2 Linear Temporal Logic and Büchi Automata

Linear Temporal Logic (LTL) consists of a set of atomic propositions, Boolean operators, and temporal operators. The syntax of LTL formulas is defined recursively over a set of atomic propositions AP as follows:

$$\varphi ::= \top \mid \alpha \in AP \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$

where φ, φ_1, and φ_2 are LTL formulas, $\top$ (true) is Boolean constant, $\wedge$ (conjunction) and $\neg$ (negation) are Boolean connectives, and $\mathbf{X}$ (*next*) and $\mathbf{U}$ (*until*) are temporal operators. These operators can be used to define additional temporal operators: *eventually*, $\mathbf{F}\varphi := \top \mathbf{U} \varphi$; and *always*, $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$.

It is known that any LTL formulas can be converted into various ω-automata, which accept an infinite word [11]. According to the definition of their acceptance condition, ω-automata can be classified into types such as Büchi automaton, Rabin automaton, among others. We first describe a transition-based generalized Büchi automaton (tGBA), followed by an introduction of the transition-based limit-deterministic generalized Büchi automaton (tLDGBA).

Definition 2. *A tGBA is a tuple* $B = (X, x_{init}, \Sigma, \delta, \mathcal{F})$*, where* X *is a finite set of states,* $x_{init} \in X$ *is the initial state,* Σ *is an input alphabet including* ε*,* $\delta \subset X \times \Sigma \times X$ *is a set of transitions, and* $\mathcal{F} = \{F_j\}_{j=1}^{n}$ *is an acceptance condition, where for each* $j \in \{1, \ldots, n\}$*,* $F_j \subset \delta$ *is a set of accepting transitions.*

An infinite sequence $r = x_0 \sigma_0 x_1 \ldots \in X(\Sigma X)^\omega$ is referred to as an infinite run if $(x_i, \sigma_i, x_{i+1}) \in \delta$ for any $i \in \mathbb{N}_0$. Here, $(\Sigma X)^\omega$ denotes an infinite sequence of repetitions of the concatenation of Σ and X, such as $\Sigma X \Sigma X \cdots$. An infinite word $w = \sigma_0 \sigma_1 \ldots \in \Sigma^\omega$ is an infinite sequence of input alphabets, and we denote a set of infinite runs generated when a tGBA B is given an infinite word w as $\mathrm{Run}(B; w)$. Furthermore, we refer to the set of transitions that occur infinitely within the run r as $\inf(r) \subseteq \delta$. An infinite word w is accepted by a tGBA B if and only if the following condition is satisfied:

$$\exists r = x_0 \sigma_0 x_1 \ldots \in \mathrm{Run}(B; w) \text{ s.t. } \inf(r) \cap F_j \neq \emptyset, \forall F_j \in \mathcal{F}.$$

Definition 3. *A tLDGBA is a tGBA where* X *can be partitioned into two disjoint subsets* $X_{initial}$ *and* X_{final} *as follows:*

- $F_j \subset X_{final} \times \Sigma \times X_{final}$, $\forall F_j \in \mathcal{F}$,
- $\left|\{(x, \sigma, x') \in \delta; \sigma \in \Sigma, x' \in X_{final}\}\right| = 1$, $\forall x \in X_{final}$,
- $\left|\{(x, \sigma, x') \in \delta; \sigma \in \Sigma, x' \in X_{initial}\}\right| = 0$, $\forall x \in X_{final}$,

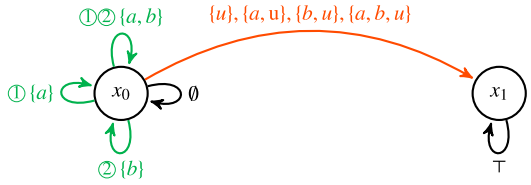

Fig. 1 tLDGBA B_φ converted from $\varphi = \mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{G}\neg u$, where $X = \{x_0, x_1\} = X_{final}$ and $x_{init} = x_0$. Green arcs labeled ① *and* ② are accepting transitions. If the input alphabet contains u, tLDGBA B_φ is no longer accepted.

- $\forall (x, \varepsilon, x') \in \delta,\ x \in X_{initial} \wedge x' \in X_{final}$.

An ε-transition occurs only in the transition from $X_{initial}$ to X_{final} with the empty string ε as the input alphabet (i.e., without reading the input alphabet). Except for this ε-transition, the transitions occurring within the states $X_{initial}$ and X_{final} are deterministic, respectively. It is known that, for any LTL formula φ, there exists the tLDGBA that accepts all words satisfying φ [11]. We denote a tLDGBA B converted from an LTL formula φ as B_φ, whose input alphabets is given by $\Sigma = 2^{AP} \cup \{\varepsilon\}$.

A simple example of an LTL formula and a tLDGBA are shown below.

Example 1. *Let the LTL formula* φ *be expressed as:*

$$\varphi = \mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{G}\neg u,$$

where a, b, u *are atomic propositions, and this LTL formula* φ *can be converted into the tLDGBA* B_φ *shown in Fig. 1. The acceptance condition of the tLDGBA* B_φ *is* $\mathcal{F} = \{F_j\}_{j=1}^{2}$*, where* $F_1 = \{(x_0, \{a\}, x_0), (x_0, \{a, b\}, x_0)\}$*,* $F_2 = \{(x_0, \{b\}, x_0), (x_0, \{a, b\}, x_0)\}$*. In Fig. 1, the green arcs labeled* ① *and* ② *represent* F_1 *and* F_2*, respectively. Once the input alphabet contains* u*, the state of the tLDGBA* B_φ *transitions to* x_1 *and it is no longer accepted.*

Furthermore, we use an automaton obtained by augmenting a tLDGBA B_φ. We denote this automaton as the augmented tLDGBA $\bar{B}_\varphi$, which accepts the same language as the tLDGBA B_φ and can ensure that acceptance transitions occur infinitely often. For details on how to augment a tLDGBA, refer to [8].

3. Proposed Multi-Agent Reinforcement Learning Methods

In this section, we propose two novel MARL methods for acquiring the control policy that satisfies a given LTL formula.

When LTL is applied to reinforcement learning, in general, the environment is firstly represented by a labeled MDP, and an LTL formula representing the control specification given to the agent is transformed into an ω-automaton [8], [12]. Then, we construct a product MDP obtained by taking the composite product of the MDP and the ω-automaton, and obtain a control strategy by reinforcement learning on this product MDP.

We utilize Q-learning [13], which is one of the most well-known reinforcement learning methods. In Q-learning, the Q-table is updated based on the (immediate) rewards the agent receives for taking actions in a given environment. The Q-table records the evaluation values of the possible actions in each state. The formula for updating each element of the Q-table (i.e., the Q value $Q(s,a)$) is given as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_a Q(s',a)\right],$$

where s is a current state, a is a selected action, s' is next state, $\alpha \in [0,1]$ is the learning rate, r is the positive reward, $\gamma \in [0,1)$ is the discount factor. The goal of Q-learning is to obtain an optimal policy $Q^*(s,a)$ that maximizes the sum of the rewards by iteratively updating the Q-table.

Note that each state transition of N agents occurs stochastically and is assumed to be synchronous, i.e., all agents either move in the intended direction or stay in the same state. To realize the synchronous move, we assume that each agent communicates with other agents. Moreover, it is expected that the learning works efficiently by the synchronous move. Furthermore, we assume that agents do not interfere with each other's actions, such as collisions.

3.1 Proposed Method 1: Simple Method

In this method, the state space is represented by a direct product, which causes the number of states and the size of the Q-table grow exponentially with the number of agents. There are cases that implementation may be difficult.

Definition 4. *Given N agents with an MDP $M_i = (S_i, A_i, s_{init,i}, P_i, AP_i, L_i)$ of agent $i \in \{1,\dots,N\}$ and an augmented tLDGBA $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$, a product MDP is a tuple $M^\otimes = M^N \otimes \bar{B}_\varphi = (N, S^\otimes, A^\otimes, s^\otimes_{init}, P^\otimes, \mathcal{F}^\otimes)$, where $M^N = M_1 \times \cdots \times M_N$. Each element of $M^\otimes$ is as follows:*

- $S^\otimes = S_1 \times \cdots \times S_N \times \bar{X}$ *: a finite set of states,*
- $A^\otimes = A_1 \times \cdots \times A_N \cup \{\varepsilon_{\bar{x}'}; \exists \bar{x}' \in X \text{ s.t. } (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}\}$ *: a finite set of actions, where $\varepsilon_{\bar{x}'}$ is a action corresponding to the ε-transition to $\bar{x}' \in \bar{X}$ on $\bar{B}_\varphi$,*
- $s^\otimes_{init} = (s_{init,1}, \dots, s_{init,N}, \bar{x}_{init})$ *: an initial state,*
- $P^\otimes : S^\otimes \times A^\otimes \times S^\otimes \to [0,1]$ *: a transition function, which is represented as follows:*

$$P^\otimes(s^{\otimes\prime}|s^\otimes, a^\otimes) = \begin{cases} p & \text{if } a^\otimes = (a_1,\dots,a_N), \left(\bar{x}, \bigcup_{i=1}^N \left(L_i((s_i,a_i,s_i'))\right), \bar{x}'\right) \\ & \quad \in \bar{\delta}, \forall i \in \{1,\dots,N\},\ a_i \in \mathcal{A}(s_i), \\ 1 & \text{if } a^\otimes = \varepsilon_{\bar{x}'}, (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}, \forall i \in \{1,\dots,N\}, s_i = s_i', \\ 0 & \text{otherwise,} \end{cases}$$

where $p \in [0,1]$, $s^\otimes = (s_1,\dots,s_N,\bar{x})$, $s^{\otimes\prime} = (s_1',\dots,s_N',\bar{x}')$, and $a^\otimes = (a_1,\dots,a_N), \varepsilon_{\bar{x}'}$.

- $\mathcal{F}^\otimes = \{F_j^\otimes\}_{j=1}^n$ *: an accepting condition, where for each $j \in \{1,\dots,n\}$, a set of accepting transitions is $F_j^\otimes = \left\{(s^\otimes, a^\otimes, s^{\otimes\prime}) \in S^\otimes \times A^\otimes \times S^\otimes;\ P^\otimes(s^{\otimes\prime}|s^\otimes, a^\otimes) > 0,\ \left(\bar{x}, \bigcup_{i=1}^N \left(L_i((s_i,a_i,s_i'))\right), \bar{x}'\right) \in \bar{F}_j\right\}$, where $s^\otimes = (s_1,\dots,s_N,\bar{x})$, $s^{\otimes\prime} = (s_1',\dots,s_N',\bar{x}')$, and $a^\otimes = (a_1,\dots,a_N), \varepsilon_{\bar{x}'}$.*

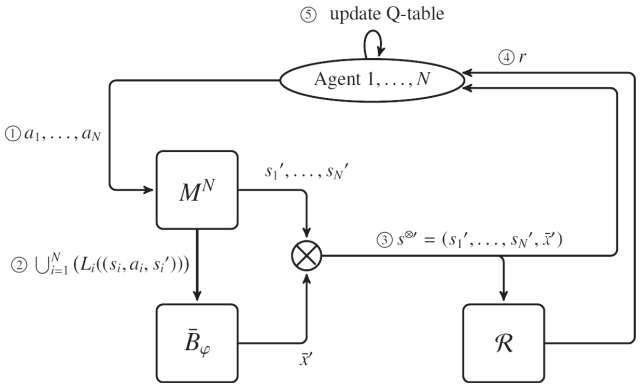

Fig. 2 Learning procedure of the simple method.

The reward function is defined as follows, where N agents are given a constant reward if an accepting transition of product MDP $M^\otimes$ occurs.

Definition 5. *Given N agents, the reward function $\mathcal{R} : S^\otimes \times A^\otimes \times S^\otimes \to \mathbb{R}_{\geq 0}$ is defined as:*

$$\mathcal{R}(s^\otimes, a^\otimes, s^{\otimes\prime}) = \begin{cases} r & \text{if } \exists j \in \{1,\dots,n\},\ (s^\otimes, a^\otimes, s^{\otimes\prime}) \in F_j^\otimes, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where r is a positive value.

In this method, the reward function is used to obtain a control strategy by reinforcement learning on a product MDP $M^\otimes$. Figure 2 shows the learning procedure of this method.

In this method, the size of a finite set of states $S^\otimes$ grows exponentially as the number of agents increases. In addition, if we denote the Q-table as $Q(s^\otimes, a^\otimes)$, its size is as follows:

$$|\, Q(s^\otimes, a^\otimes)\,| = |\, S_1 \times \cdots \times S_N \times \bar{X}\,| \cdot |\, A_1 \times \cdots \times A_N\,|, \tag{2}$$

and it also grows exponentially as the number of agents increases.

3.2 Proposed Method 2: Distributed Method

In order to solve the technical issue of the simple method, consider introducing an aggregator to partition the state space and the Q-table. Note that $\bar{B}_\varphi$ is the same among agents, which represents their common goal.

Definition 6. *Given N agents with an MDP $M_i = (S_i, A_i, s_{init,i}, P_i, AP_i, L_i)$ of agent $i \in \{1,\dots,N\}$ and an augmented tLDGBA $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$, a product MDP of agent i is a tuple $M_i^\otimes = M_i \otimes \bar{B}_\varphi = (S_i^\otimes, A_i^\otimes, s^\otimes_{init,i}, P_i^\otimes, \mathcal{F}_i^\otimes)$. Each element of $M_i^\otimes$ is as follows:*

- $S_i^\otimes = S_i \times \bar{X}$ *: a finite set of states,*
- $A_i^\otimes = A_i \cup \{\varepsilon_{\bar{x}'}; \exists \bar{x}' \in X \text{ s.t. } (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}\}$ *: a finite set*

of actions, where $\varepsilon_{\bar{x}'}$ is a action corresponding to the ε-transition to $\bar{x}' \in \bar{X}$ on $\bar{B}_\varphi$,

- $s^{\otimes}_{init,i} = (s_{init,i}, \bar{x}_{init})$ *: an initial state,*
- $P^{\otimes}_i : S^{\otimes}_1 \times \cdots \times S^{\otimes}_N \times A^{\otimes}_1 \times \cdots \times A^{\otimes}_N \times S^{\otimes}_i \to [0,1]$ *: a transition function, which is represented as follows:*

$$P^{\otimes}_i(s^{\otimes\prime}_i | s^{\otimes}_1, \ldots, s^{\otimes}_N, a^{\otimes}_1, \ldots, a^{\otimes}_N) = \begin{cases} P_i(s_i' | s_i, a_i) & \text{if } \left(\bar{x}, \bigcup_{i=1}^{N} \left(L_i((s_i, a_i, s_i'))\right), \bar{x}'\right) \in \bar{\delta}, \\ & \quad \forall i \in \{1, \ldots, N\},\ a_i \in \mathcal{A}(s_i), \\ 1 & \text{if } s_i = s_i',\ (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}, \\ & \quad \forall i \in \{1, \ldots, N\},\ a^{\otimes}_i = \varepsilon_{\bar{x}'}, \\ 0 & \text{otherwise}, \end{cases}$$

where for each $i \in \{1, \ldots, N\}$, $s^{\otimes}_i = (s_i, \bar{x}), s^{\otimes\prime}_i = (s_i', \bar{x}')$, and $a^{\otimes}_i = a_i, \varepsilon_{\bar{x}'}$.

- $\mathcal{F}^{\otimes}_i = \{F^{\otimes}_{i,j}\}^{n}_{j=1}$ *: an accepting condition, where for each $j \in \{1, \ldots, n\}$, a set of accepting transitions is $F^{\otimes}_{i,j} = \{(s^{\otimes}_i, a^{\otimes}_i, s_i{}^{\otimes\prime}) \in S^{\otimes}_i \times A^{\otimes}_i \times S^{\otimes}_i;\ P^{\otimes}_i(s^{\otimes\prime}_i \mid s^{\otimes}_1, \ldots, s^{\otimes}_N, a^{\otimes}_1, \ldots, a^{\otimes}_N) > 0, \left(\bar{x}, \bigcup_{i=1}^{N} \left(L_i((s_i, a_i, s_i'))\right), \bar{x}'\right) \in \bar{F}_j\}$. where for each $i \in \{1, \ldots, N\}$, $s^{\otimes}_i = (s_i, \bar{x}), s^{\otimes\prime}_i = (s_i', \bar{x}')$, and $a^{\otimes}_i = a_i, \varepsilon_{\bar{x}'}$.*

The reward function is defined as a function that gives each agent a constant reward when at least one agent causes an accepting transition.

Definition 7. *Given N agents, the reward function $\mathcal{R} : S^{\otimes}_1 \times \cdots \times S^{\otimes}_N \times A^{\otimes}_1 \times \cdots \times A^{\otimes}_N \times S^{\otimes}_1 \times \cdots \times S^{\otimes}_N \to \mathbb{R}^{N}_{\geq 0}$ is defined as:*

$$\mathcal{R}(s^{\otimes}_1, \ldots, s^{\otimes}_N, a^{\otimes}_1, \ldots, a^{\otimes}_N, s^{\otimes\prime}_1, \ldots, s^{\otimes\prime}_N) = \begin{cases} (r_1, \ldots, r_N) & \text{if } \exists i \in \{1, \ldots, N\}, \exists j \in \{1, \ldots, n\}, \\ & \quad (s^{\otimes}_i, a^{\otimes}_i, s^{\otimes\prime}_i) \in \bar{F}^{\otimes}_{i,j}, \\ (0, \ldots, 0) & \text{otherwise}, \end{cases} \quad (3)$$

where $r_1, \ldots, r_N$ is a positive value.

In this method, we introduce an aggregator that calculates how well the control specification is satisfied from agent's actions. The learning procedure with an aggregator is described below.

Learning Procedure of Distributed Method:

Step 1: Each agent chooses an action a_i in the current state on its MDP M_i based on its own policy.

Step 2: The next state s_i' is determined on M_i, where each agent's transition occurs synchronously due to communication among the agents.

Step 3: The labeling function L_i outputs the label.

Step 4: The aggregator collects the union set of labels of each agent and calculates the state transition of the augmented tLDGBA $\bar{B}_\varphi$.

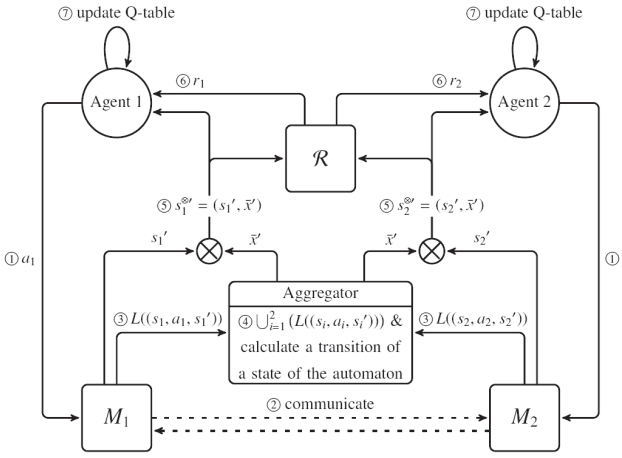

Fig. 3 Learning procedure of the distributed method for $N = 2$ agents. The introduction of an aggregator enables the partitioning of both the MDP and the Q-table.

Step 5: The next state $s^{\otimes\prime}_i$ on the product MDP $M^{\otimes}_i$ is determined by combining $\bar{x}'$ calculated by the aggregator and s_i'. Then, $s^{\otimes\prime}_i$ is given to the reward function $\mathcal{R}$ and each agent.

Step 6: The reward function $\mathcal{R}$ calculates and distributes each agent's reward r_i.

Step 7: Each agent's Q-table is updated. Return to Step 1 and repeat the above procedure.

Figure 3 shows the learning procedure of this method in the case of two agents ($N = 2$). ①, …, ⑦ in Fig. 3 correspond to the same number of steps in the above learning procedure. In this learning procedure, the aggregator has the role of calculating the state of tLDGBA for computing the reward values. By introducing the aggregator, each agent can learn without having to observe the state of other agents.

Note that each agent has a Q-table in this method, and we denote a Q-table of agent i as $Q_i(s^{\otimes}_i, a^{\otimes}_i)$. The total of the sizes of Q-tables of N agents is as follows:

$$\sum_{i=1}^{N} \mid Q_i(s^{\otimes}_i, a^{\otimes}_i) \mid = \sum_{i=1}^{N} \mid S_i \times \bar{X} \mid \cdot \mid A_i \mid, \quad (4)$$

which grows linearly as the number of agents increases. In addition, the total number of states of product MDPs of N agents also grows linearly. Thus, the distributed method reduces both the total number of states of product MDPs and the total sizes of the Q-tables from an exponential increase to a linear increase compared to the simple method. It is expected that the learning efficiency is improved by reducing the total sizes of the Q-tables.

4. Simulation Experiment

In this section, we apply each proposed method to the surveillance problem for three robots ($N = 3$). Consider the 7×7 grid world shown in Fig. 4 as the environment given to each

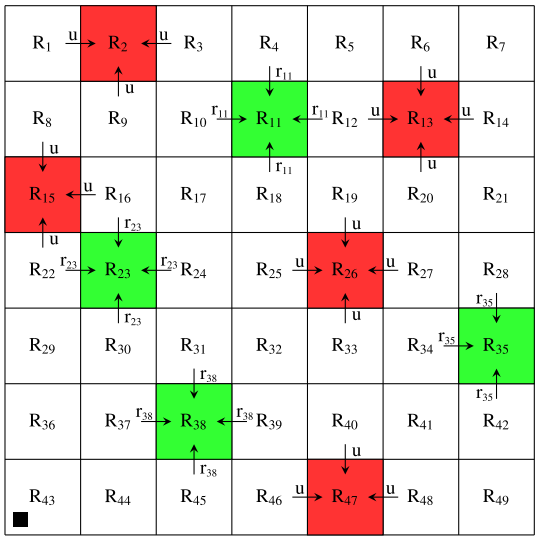

Fig. 4 7×7 grid world where $N = 3$ robots patrol. Green rooms are the states that we want agents to try to visit, while red rooms are the states that we want agents to avoid visiting. R_{43} is the initial state of all agents.

robot. The finite set of states is R_i ($i = 1, \cdots, 49$), the initial state is R_{43} and a finite set of actions is {Up, Down, Right, Left}. The robots move toward the intended state with probability 0.9 and stay in the same state with probability 0.1. The stop action is not included in the finite set of actions of the robots, because it is implicitly included in the "stay" of the transition probability function. In addition, the finite set of atomic propositions is $\{r_{11}, r_{23}, r_{35}, r_{38}, u\}$, and the labeling function is as follows:

$$L\big((s,a,s')\big) = \begin{cases} \{r_{11}\} & \text{if } s' = R_{11}, \\ \{r_{23}\} & \text{if } s' = R_{23}, \\ \{r_{35}\} & \text{if } s' = R_{35}, \\ \{r_{28}\} & \text{if } s' = R_{38}, \\ \{u\} & \text{if } s' = R_i,\ i = \{2, 13, 15, 26, 47\}, \\ \emptyset & \text{otherwise.} \end{cases}$$

In this problem, the control specification given to the robots is "to visit the rooms R_i ($i = 11, 23, 35, 38$) infinitely often, while to avoid visiting the rooms R_i ($i = 2, 13, 15, 26, 47$)". This can be described as the following LTL formula φ:

$$\varphi = \mathbf{GF}r_{11} \wedge \mathbf{GF}r_{23} \wedge \mathbf{GF}r_{35} \wedge \mathbf{GF}r_{38} \wedge \mathbf{G}\neg u,$$

where r_i ($i = 11, 23, 35, 38$) are atomic propositions that mean to visit the rooms R_i ($i = 11, 23, 35, 38$) and u is one that means to visit the rooms R_i ($i = 2, 13, 15, 26, 47$). The number of states in the augmented tLDGBA is derived as $| \bar{X} | = 30$.

For the simple method, from (2), the number of states of the product MDP is

$$| S^{\otimes} | = | S_1 \times S_2 \times S_3 | \cdot | \bar{X} | = 49^3 \cdot 30 = 3529470$$

and the size of the Q-table is

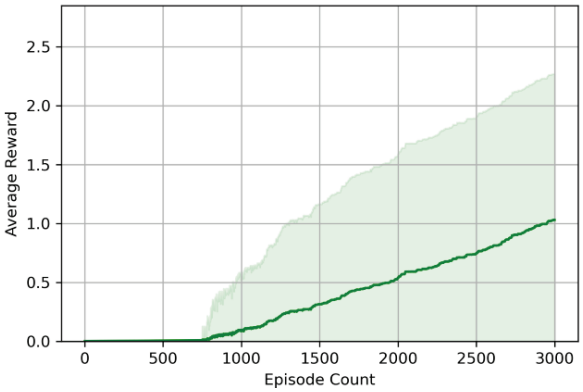

Fig. 5 Reward history for the simple method. The mean of the average reward for each episode was calculated across 500 learning sessions. The green areas indicate the range of standard deviations.

$$| Q(s^{\otimes}, a_1, a_2, a_3) | = | S^{\otimes} | \cdot | A_1 \times A_2 \times A_3 | \\ = 3529470 \cdot 4^3 = 225886080.$$

On the other hand, for the distributed method, from (4), the total number of states of the three agents' product MDPs is

$$\sum_{i=1}^{3} | S_i^{\otimes} | = \sum_{i=1}^{3} (| S_i | \cdot | \bar{X} |) \\ = 3 \cdot (49 \cdot 30) = 3 \cdot 1470 = 4410,$$

and the total size of the Q-tables is

$$\sum_{i=1}^{3} | Q_i(s_i^{\otimes}, a_i) | = \sum_{i=1}^{3} (| S_i^{\otimes} | \cdot | A_i |) \\ = 3 \cdot (1470 \cdot 4) = 17640.$$

It was confirmed that the distributed method reduces both the number of states of the product MDP and the size of the Q-table.

We ran a total of 500 learning sessions of 3000 episodes, with 2000 steps in each episode. Here, the reward design was set to "all agents get the same reward if even one agent's transition satisfies the acceptance condition". The reward r in (1) is set as $r = 2$. The reward r_i in (3) is set as $r_i = 2$. The discount rate and the learning rate are set as $\gamma = 0.99$ and $\alpha = 1/(1 + ep)^{0.5}$, respectively, where ep was the number of episodes.

Figures 5 and 6 show the reward history obtained by each method. In both figures, the green areas indicate the range of standard deviations. Moreover, Figs. 7 and 8 show representative three agents' trajectories resulted from the trainings of each method, where the circles in both figures represent accepting transitions. For the simple method, as shown in Fig. 7, we see that in about one-third of the cases, the learning was not successful and the appropriate control policy was not achieved. On the other hand, for the distributed method, as shown in Fig. 8, we see that each agent cooperatively repeats each accepting transition, and the agents cannot solve the task only about one-hundredth of

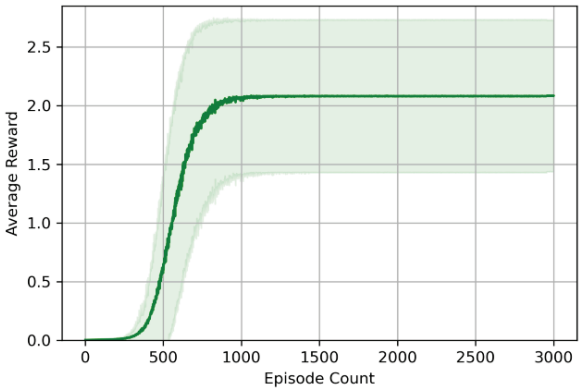

Fig. 6 Reward history for the distributed method. The mean of the average reward for each episode was calculated across 500 learning sessions. The green areas indicate the range of standard deviations.

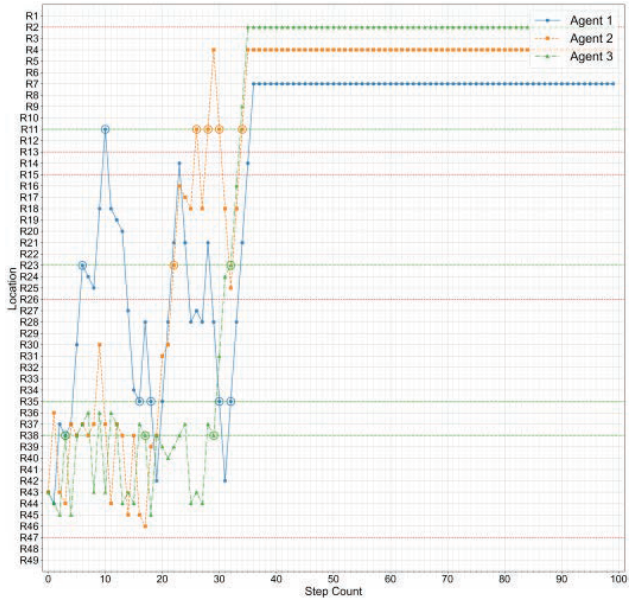

Fig. 7 Three agents' trajectories for the simple method. The circles indicate accepting transitions. One case where learning was not successful and an appropriate control policy was not achieved.

cases. Thus, the performance of the distributed method is better than the simple method. The mean and standard deviation of the computation time was 5413 (±1051) seconds for the simple method and 2812 (±360) seconds for the distributed method, where the simulations were run in Google Colaboratory. By comparing Fig. 5 with Fig. 6 and the computation times of each method, the distributed method performed better in terms of the higher earned rewards, the faster convergence and the shorter computational time.

5. Conclusion and Future Work

In this paper, we proposed two novel MARL methods (the simple method and the distributed method) with the control specification described by LTL formulas. The distributed method with an aggregator reduces linearly the growth of

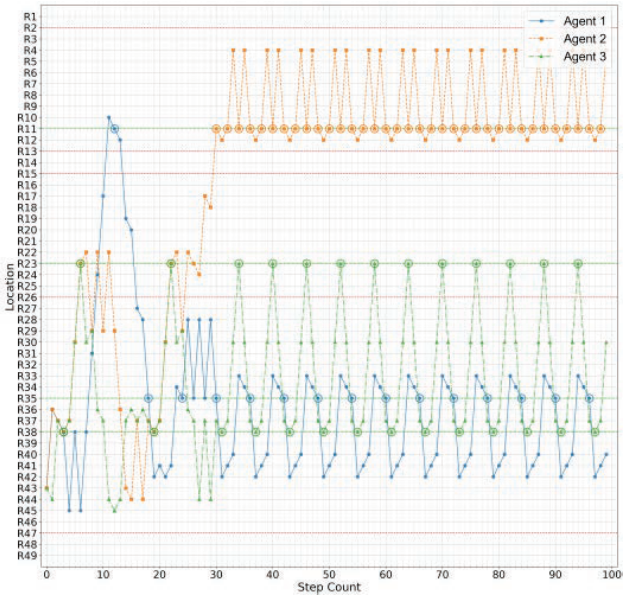

Fig. 8 Three agents' trajectories for the distributed method. The circles indicate accepting transitions. It can be seen that each agent cooperatively repeats each accepting transition.

the number of states and the size of the Q-table, which occur with the increase of the number of agents in the simple method. It is shown that the distributed method improves the rewards and reduces the computation time.

Future work includes further improving the performance of the proposed methods. This also includes addressing a reward distribution problem and learning methods that reduce the frequency of communication by the aggregator in the distributed method.

This work was partly supported by JSPS KAKENHI Grant Numbers JP19H02158, JP21H04558, JP22K04163.

References

[1] G. Weiss, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, 1999.
[2] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[3] C. Baier and J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
[4] B.G. León and F. Belardinelli, "Extended Markov games to learn multiple tasks in multi-agent reinforcement learning," arXiv preprint arXiv:2002.06000, 2020.
[5] C. Sun, X. Li, and C.A. Belta, "Automata guided semi-decentralized multi-agent reinforcement learning," 2020 American Control Conference, pp.3900–3905, 2020.
[6] D. Muniraj, K.G. Vamvoudakis, and M. Farhood, "Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep Q-learning approach," 2018 IEEE Conference on Decision and Control, pp.4141–4146, 2018.
[7] L. Hammond, A. Abate, J. Gutierrez, and M. Wooldridge, "Multi-agent reinforcement learning with temporal logic specifications," 20th International Conference on Autonomous Agents and MultiAgent Systems, pp.583–592, 2021.
[8] R. Oura, A. Sakakibara, and T. Ushio, "Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized Büchi automata," IEEE Control Syst. Lett.,

vol.4, no.3, pp.761–766, 2020.
[9] J. Foerster, I.A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," Advances in Neural Information Processing Systems, 29, 2016.
[10] R. Lowe, Y. Wu, A. Tamar, J. Harb, OpenAI P. Abbeel, and I. Mordatch. "Multi-agent actor-critic for mixed cooperative-competitive environments," Advances in Neural Information Processing Systems, 30, 2017.
[11] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ, "Limit-deterministic Büchi automata for linear temporal logic," 28th International Conference on Computer Aided Verification, pp.312–332, Springer, 2016.
[12] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained reinforcement learning," arXiv preprint arXiv:1801.08099, 2018.
[13] C.J.C.H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol.8, pp.279–292, 1992.

Keita Terashima received the B.E. degree in 2022 from Hokkaido University. He is currently a master course student at the Graduate School of Information Science and Technology, Hokkaido University. His research interests include multi-agent reinforcement learning and temporal logic.

Koichi Kobayashi received the B.E. degree in 1998 and the M.E. degree in 2000 from Hosei University, and the D.E. degree in 2007 from Tokyo Institute of Technology. From 2000 to 2004, he worked at Nippon Steel Corporation. From 2007 to 2015, he was an Assistant Professor at Japan Advanced Institute of Science and Technology. From 2015 to 2022, he was an Associate Professor at Hokkaido University. Since 2023, he has been a Professor at the Faculty of Information Science and Technology, Hokkaido University. His research interests include discrete event and hybrid systems. He is a member of IEEE, IEEJ, IEICE, ISCIE, and SICE.

Yuh Yamashita received his B.E., M.E., and Ph.D. degrees from Hokkaido University, Japan, in 1984, 1986, and 1993, respectively. In 1988, he joined the faculty of Hokkaido University. From 1996 to 2004, he was an Associate Professor at the Nara Institute of Science and Technology, Japan. Since 2004, he has been a Professor of the Graduate School of Information Science and Technology, at Hokkaido University. His research interests include nonlinear control and nonlinear dynamical systems. He is a member of SICE, ISCIE, IEICE, RSJ, and IEEE.