# Learning Method by Sharing Activity Histories in Multiagent Environment

Keinosuke Matsumoto, Takuya Gohara, and Naoki Mori
Department of Computer Science and Intelligent Systems
Graduate School of Engineering, Osaka Prefecture University
Sakai, Osaka, Japan
email: {matsu, gohara, mori}@cs.osakafu-u.ac.jp

*Abstract*—**Applications of multiagent systems are expected for parallel and distributed processing. Reinforcement learning is used as an implementation method for learning the actions of the agent. However, when systems must control many agents, the speed of learning becomes slower. Hence, Modular Q-Learning is proposed to solve this problem. Given that it deals with partial states, the number of states is reduced to avoid exponential increases. However, if $n$ agents exist, they need $n$ ($n$−1) learning tables, and therefore require a lot of memory. To solve the problem, Centralized Modular Q-Learning is proposed. In this method, the agent has only one learning table. Given that agents do not distinguish other agents, the number of learning tables is reduced. This study improves these methods and proposes a new reinforcement learning method that can learn quickly by using the past actions of its own and other agents. The proposed method can learn good actions in fewer trials. However, if agents continuously learn, the learning efficiency will deteriorate. The method reduces the effects of the actions of other agents in the late stage of learning. Therefore, agents are able to learn suitable actions. In experiments, agents are able to find a good strategy in a small number of trials than the conventional methods. In addition, agents learn actions in hunter games in various environments. The results show that the proposed method is an efficient reinforcement learning method.**

***Keywords—machine learning; Q-learning; sharing of activity histories; agents; hunter game***

## I. INTRODUCTION

This paper is based on the study [1] presented at the ADVCOMP 2016. In recent years, information has distributed and increased largely due the rapid development of the Internet and multimedia. Systems also become larger and complicated. It is difficult for centralized systems, which judge by bringing information in one place, to deal with a lot of information and process it. From the viewpoint of parallel and distributed processing, the application of multiagent systems [2] that exchange information between agents [3] is expected.

It is difficult to follow environmental changes that humans could not forecast beforehand, and they do not carry out suitable actions. It is most important for each agent in a multiagent system to learn by itself. Each agent needs to learn a suitable judgment standard from its experience and information collected from other agents. Reinforcement learning [4][5] attracts attention as an implementation method for multiagent systems. It can be very effective means because it autonomously learns by setting the only reward, if a goal has been given.

A hunter game [6] is widely used as a cooperative problem solving [7][8] under multiagent environment as a benchmark of reinforcement learning. If a hunter game becomes complicated and the number of agents increases, the number of states increases exponentially. The speed of learning slows down. Ono et al. proposed Modular Q-Learning (MQL) [9] to solve this problem, but it had a disadvantage of using a lot of memory. Knowledge sharing methods [10][11][12][13] were also proposed. Reference [12] needs to build a tree structure model and [13] consumes a considerable amount of memory to store auxiliary variables that are used to record the trajectory of states, action, and rewards. With respect to memory, another method that reduced memory [14] was proposed. In this method, each agent has only one Q-value table by not distinguishing each agent with the same purpose.

Based on these methods, this paper proposes a new method that increases learning efficiency by using each agent's activity history of hunter agents. The method does not need preparation of any special model or communication algorithms between agents, strategies to exchange information [15][16], special exploration agents [17][18][19] and others according to various situations. This method saves only activity histories and updates the Q-value using its own or other hunters' activity histories. In this manner, the method shares experiences between agents simply by adding other hunters' activity histories to the Q-value table and picks up learning speed, which makes collective intelligence efficient.

This paper is organized as follows. In Section II, the explosion of the number of states in the reinforcement learning is explained. In Section III, conventional methods are described. In Section IV, the proposed method is explained. In Section V, the results of application experiments confirm the validity of the proposed method. Finally, in Section VI, the conclusion and future work are presented.

## II. HUNTER GAME

This section describes a hunter game and the explosion of the number of states.

### A. Definition of Hunter Game

A hunter game is one of the standard problems in multiagent systems. It is a game where multiple hunters catch a prey (runaway) by chasing in a two-dimensional field. The definition of a hunter game in this study is shown below.

-A field is a two-dimensional lattice and torus space as shown in Fig. 1.

-It is possible for multiple agents to take one lattice space.

-Each agent can take five actions of moving, such as right, left, up, down or stop.

-A hunter has perfect perception, and it recognizes a prey and other hunters in relative coordinates from itself.

-A unit of time that each agent takes one action is called a time step, and a period from an initial state to a goal (i.e., hunters catch a prey) is called an episode.

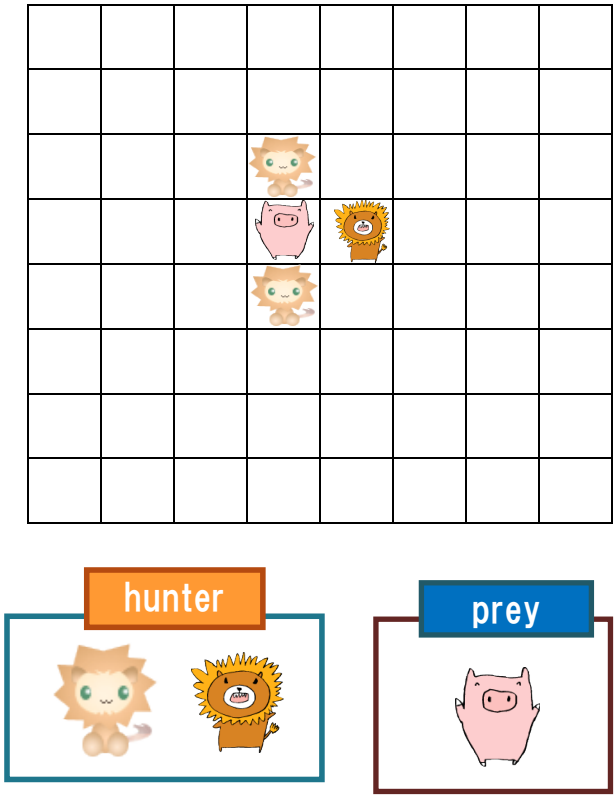


Figure 1. Hunter game.

TABLE I. Number of states $m^{2n}$.

| $n \backslash m$ | 3 | 5 | 7 | 9 |
|---|---|---|---|---|
| 1 | 9 | 25 | 49 | 81 |
| 2 | 81 | 625 | 2401 | 6541 |
| 3 | 729 | 15025 | 117649 | 531441 |
| 4 | 6561 | 390625 | 5764801 | 43046721 |

### B. Explosion of the Number of States

Q-Learning [20] is one of the bootstrap-type reinforcement learning. In Markov decision process, which is similar to Q-Learning, if the learning rate is appropriately adjusted, convergence to an optimal solution in infinite time has been proven [21].

In Q-Learning of the hunter game, an action is evaluated on a pair ($s$, $a$) considering all observable states $S$ ($S \ni s$) and possible actions $A$ ($A \ni a$). The evaluated value is utilized for the same pair of state and action. It requires a lot of information on ($s$, $a$) to make Q-Learning effective. For example, if the size of the field is $m \times m$ and the number of hunters is $n$, one hunter can see $m^{2n}$ identifiable states (positional combinations of other hunters and prey). Table I shows the number of states for each number of hunters $n$ and field size $m$. Given that each state has five kinds of actions, the state and action pair is $5m^{2n}$.

In the hunter game with multiple hunters, state explosion cannot be avoided because the exponent includes $n$. The explosion of the number of states results in slower learning speed. Therefore, in Q-Learning in multiagent environment, how the number of states is reduced is an important subject.

## III. CONVENTIONAL METHODS

This section describes related work of this study.

### A. Modular Q-Learning

Ono et al. [9] proposed MQL to solve the state explosion in hunter games. Completely Perceptual Q-Learning (CPQL) [22] is a perfect perception learning, and it uses relative coordinates of all hunters to define states. Moreover, MQL uses a partial state that consists of a hunter and another one. The number of states of field size $m \times m$ and $n$ hunters is $m^4$. Given that the exponent is a constant and is not influenced by the number of hunters, it can prevent the state explosion.

Learning accuracy of MQL deteriorates because of imperfect perception by the observing partial states. In addition, if $n$ hunters exist, the number of partial states becomes $n$-1, and $n$-1 learning machines are prepared per hunter. A total of $n(n$-1) learning machines are needed. The size of the Q-value tables tends to increase, and the amount of memory will increase.

### B. Centralized Modular Q-Learning

Matsumoto et al. [14] proposed a Centralized Modular Q-Learning (CMQL) to solve the memory problem of MQL. In a hunter game, hunters should just surround a prey. It is not necessary to recognize the kind of hunters that surround the prey. Therefore, CMQL does not distinguish the characteristics of each hunter, and $n$-1 learning machines that

the hunter has in MQL can be reduced to one learning machine. In CMQL, a hunter has only one Q-value table of the partial state. Given that the number of Q-value tables becomes one per hunter, only $n$ Q-value tables are required in all if $n$ hunters exist.

The number of states will increase, and the speed of learning will become slow in the hunter games of three or more hunters. To solve this problem, CMQL is introduced. CMQL improves the degradation of learning by parallel and switching learning [22]. The increase of memory is reduced by one learning machine per hunter.

*1) Parallel learning*

Imperfect perception learning is used to make learning quicker in the early stage and to accelerate learning processes to some extent. We switch to a perfect perception at a time. Owing to imperfect perception, the learning accuracy of CMQL is lowered, and the action selection will change for the worse in the later stages. A long-term performance is inferior compared with CPQL. The influence of lowered accuracy in early stages does not disappear, and the accuracy of action selection cannot be kept perfect.

Parallel learning is a method of using CMQL that excels in early short-term learning and CPQL that excels in long-term learning simultaneously. A decent action series is found by CMQL, and it is made to converge, where further convergence is expected by switching to CPQL at a suitable time. To obtain the suitable switching time for parallel learning, the mean unlearning entropy is defined. The probability P($s$, $a$), which chooses action $a$ at the time of state $s$ and the unlearning entropy I($s$) are shown below. I is an average of I($s$), which is averaged for all the states contained in episode E and all agents.

$$\mathrm{P}(s,a) = \frac{Q(s,a)}{\sum_{i \in \mathrm{A}} Q(s,i)} \tag{1}$$

$$\mathrm{I}(s) = -\frac{1}{\log n_a} \sum_{a} \mathrm{P}(s,a) \log \mathrm{P}(s,a) \tag{2}$$

$$\mathrm{I} = \frac{1}{n_p |\mathrm{E}|} \sum_{s \in \mathrm{E}} \mathrm{I}(s) \tag{3}$$

where $Q$ ($s$, $a$) is the Q-value of action $a$ in state $s$, $A$ is a set of all possible actions, $n_a$ is the number of actions that can be chosen, $n_p$ is the number of hunters, and |E| is the number of states contained in episode E. I comes close to 0 when the learning progresses. Moreover, it is 1 if no learning is carried on.

*2) Switching learning method*

If parallel learning of CMQL and CPQL are used at the same time, the amount of memory will increase because the two learning methods must use a lot of memory. Before switching learning, only the learning machine of CMQL is in the memory; and after switching, only the learning machine of CPQL is in the memory. By this process, learning can always be carried out under the memory of CPQL before and after switching.

The delivery technique of Q-value at the time of switching is shown: Three hunters ($s_1$, $s_2$, $s_3$) exist with states of $s_1(x_1, y_1)$, $s_2(x_2, y_2)$, and $s_3(x_3, y_3)$. Hunter $s_1$'s Q-value of CMQL is $Q_m$ ($s_1$, $T$, $a$). Moreover, the Q-value of CPQL is $Q_c$ ($s_1$, $s_2$, $s_3$, $a$). Where $T$ is a state of another hunter, and $a$ is one of the actions. The $Q$-value cannot be copied easily because the expression forms are different. Q-value is delivered in the following formula:

$$Q_c[x_1][y_1][x_2][y_2][x_3][y_3][a] = \frac{Q_m[x_1][y_1][x_2][y_2][a] + Q_m[x_1][y_1][x_3][y_3][a]}{2} \tag{4}$$

This formula can deliver the same Q-value to all combinations from CMQL to CPQL. The difference between both expression forms is absorbed in this manner.

*3) Preliminary experiments*

Some preliminary experiments have been conducted to investigate the influence of mean unlearning entropy on learning. The problems that the preliminary experiments deal with are shown below. Each hunter carries out Q-learning individually, and a prey acts at random without learning in hunter games. The number of hunters is three, the field size is 7 × 7, and the cost per one-time step is 0.05. Q-value may become zero or less. To prevent this case, $\delta$ is defined as follows, and $\delta$ is added to $Q$ ($s$, $a$).

$$\delta = |\min_{a} Q(s,a)| + 0.01 \tag{5}$$

In both CPQL and CMQL, learning and discount rates are set to 0.5. Thresholds of mean unlearning entropy are set to 0.500, 0.840, and 0.947. These values correspond to switching times at 45,000, 15,000, and 3000 episodes. The resulting graphs are shown in Fig. 2. Every plot shows the average time steps to catch a prey of every 300 episodes. Given that the mean unlearning entropy comes close to 0 as the learning progresses as described in sub-section *1)*, the larger the thresholds are, the quicker it switches in the early stages of learning.

When it is switched at threshold 0.500, the number of steps to catch a prey has leaped up abruptly at the time of switching. Furthermore, it has also converged on the number of steps worse than that of CPQL. This means that the action patterns learned by CMQL is delivered to CPQL, but the deteriorated action patterns are not corrected. When the threshold is 0.840, it switches earlier than that of the threshold 0.500, but it switches similarly and the number of steps to catch a prey has leaped up abruptly. The convergent number of steps to catch is almost the same as that of CPQL. When it switches at the threshold 0.947, it has switched just before the learning accuracy of CMQL deteriorates. Change of learning machines can be performed, and the number of steps does not leap up abruptly. After the switching, the

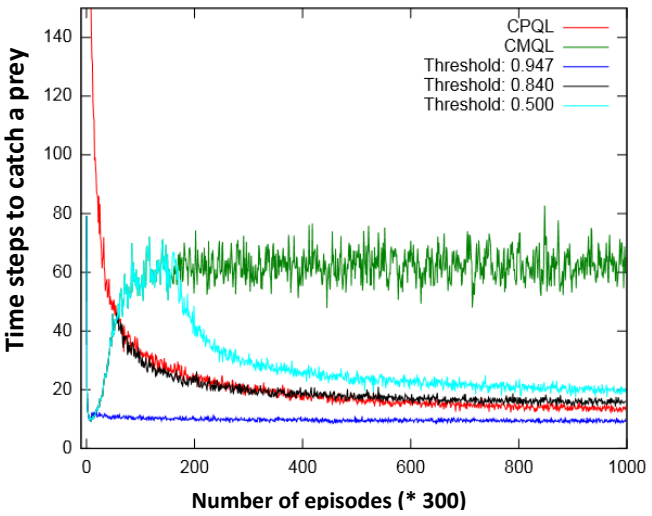


Figure 2. Learning graphs for various thresholds.

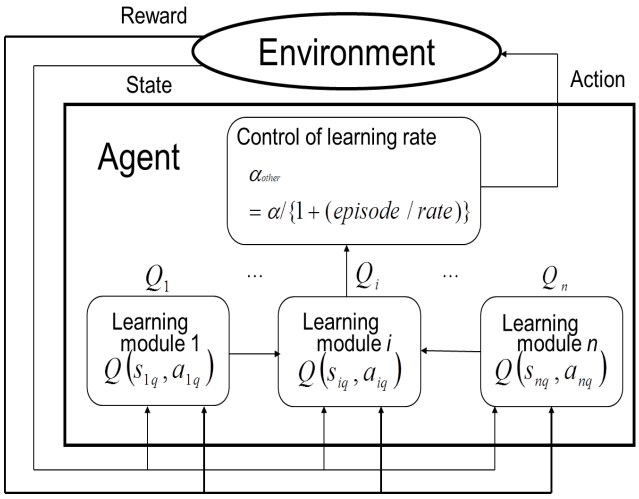


Figure 3. Architecture of the proposed method.

number of steps converges better than that of CPQL.

These results show that CMQL obtained a fewer steps solution and fewer amount of memory than those of CPQL when it switched at the threshold 0.947. Therefore, the efficiency of reinforcement learning in hunter games can be increased by CMQL. However, if the learning rates and discount rates are changed, the results will change. The inability to determine automatically an optimum switching time is a problem.

## IV. Proposed Method

In this section, a method that raises learning efficiency is described based on MQL and CMQL. Fig. 3 shows the basic concept of the proposed method. The method for defining the partial state of CMQL in a hunter game in a maze environment is examined.

### A. Redefinition of the Partial State by the Relative Coordinate Change

In MQL and CMQL, the definition of perceptual information is made by a relative coordinate from the prey. This study uses the relative coordinate from each hunter. If the coordinates of other hunters $s_1$: $(x_1, y_1)$, $s_2$: $(x_2, y_2)$, and the prey $s_p$: $(x_p, y_p)$, the partial state of the method is $<s_p, s_1>$, $<s_p, s_2>$. Therefore, perceptual information of some hunters can be constituted based on information equal to the partial states of CMQL.

### B. Perception Method of Walls

Conventional CMQL constitutes partial states based on hunter and prey coordinates. It is necessary to consider the walls in a hunter game in a maze environment. An effect seems to come out in learning results by way of defining the partial states. Given that the positions of the walls do not change in this study, all walls are grasped by the absolute coordinate system. The partial states that consider the walls using this absolute coordinates are constructed. The walls are blocks where each agent could not go through.

### C. Explorative Experiment

Two kinds of partial states are considered in this experiment. In hunter games, one partial state consists of a pair of any hunter and prey, and another pair of a wall and prey (Method 1). Another partial state is considering a hunter, prey, and wall at the same time (Method 2). Figures 4 and 5 show partial states for Methods 1 and 2, respectively. The number of states of Method 2 is larger than that of Method 1, but their memory consumption is equal. Experimental conditions were as follows:

- Size of field: 12 × 12
- Number of walls in the mazy field: 43
- Number of hunters: $n = 3$
- Action selection strategy: ε-greedy (ε = 0.01)
- Prey's action: random action
- Capture state: Four lattices in left, right, top, and bottom of a prey's position are surrounded by hunters or walls.
- Cost per one time step: 0.05
- Learning rate: α = 0.2
- Discount rate: γ = 0.8
- Maximum number of learning episodes: 300000

The comparison results are shown in Fig. 6. The number of steps of Method 2 to catch the prey decreases when the learning advances. Good learning is possible. Comparing Method 1 with Method 2, the learning speed of Method 1 is rapid, but the number of steps to catch increases. The positions of walls are considered in all partial states of Method 2. Method 2 can choose an action that bypasses walls between the hunter and prey. Some partial states of Method 1 disregard the walls, and Method 1 could not choose a good action that accesses the prey. If an element to be newly considered in the environment increases, each partial state seems to have to consider the new element at the same time.

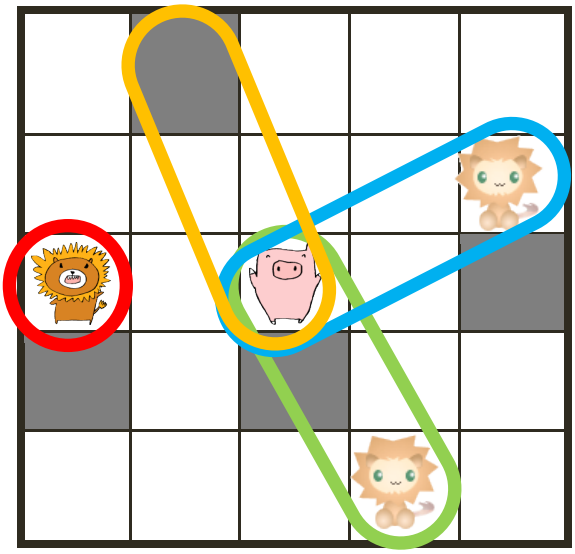

Figure 4. Partial states for Method 1.

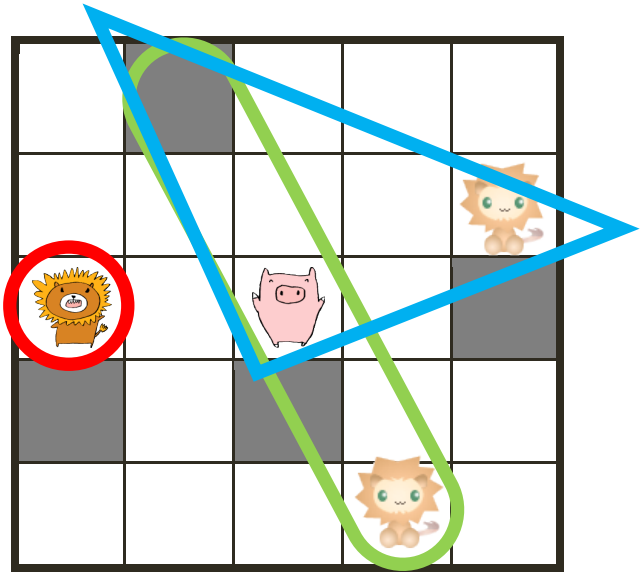

Figure 5. Partial states for Method 2.

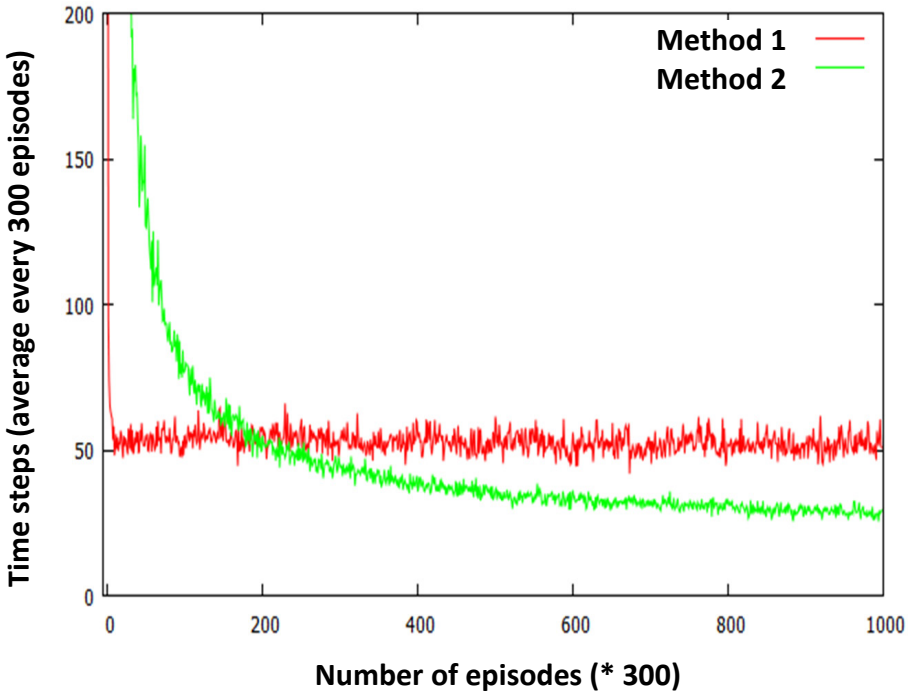


Figure 6. Results of the proposed method for maze task.

### D. Learning Method by Sharing Activity Histories

In the hunter game, all hunters have the common purpose of catching the prey. In this environment, the learned actions of other hunters to catch the prey are useful. Appropriate actions can be learned with fewer trials by learning actions of other hunters. In this study, a method of updating the Q-value based on other hunters' activity histories is proposed. The number of times of updating for every episode increases, but the method raises the learning efficiency for every episode. The algorithm of the proposed method is shown below.

The number of hunters is $n$ and a prey is caught at $q$ steps. Each hunter is observing states $s_1$, $s_2$, ---, $s_n$ and actions are $a_1$, $a_2$, ---, $a_n$.

(1) In each episode, save the hunters' coordinates and actions for every step, and for up to $t$ steps. These are activity histories.
(2) Give awards to all hunters' Q-value Q ($s_1$, $a_1$), Q ($s_2$, $a_2$), ---, Q($s_n$, $a_n$) if the prey is caught.
(3) $i = q$
(4) $Q(s_{i-1}, a_{i-1}) \leftarrow (1-\alpha)\, Q(s_{i-1}, a_{i-1}) + \alpha\, [r + \gamma \max_a Q(s_i, a_i)]$
(5) Replace $i$ by $i$-1 and repeat (4) until $i \leq q$-$t$ or $i \leq 1$.

In the algorithm mentioned above, $t$ is $t$ = 1000. Combining this algorithm with CMQL makes a more efficient learning method. Parameter $t$ is determined by the complexity of the applied problems to cover almost all states at the initial setting [23][24][25][26].

Although learning has become early in the proposed method, final learning results tend to deteriorate compared with the conventional methods without sharing activity histories. The learning accuracy of the proposed method becomes worse by learning actions of other hunters at the final learning stage. For this reason, the learning rate using other hunters' actions is decreased according to the number of episodes. Influence on learning by other hunters' actions is lessened as learning progresses. This will be an approach that utilizes other hunters' activity histories at the early learning stages and uses only each hunter's history at the final learning stage.

### E. Control of Learning Rate

It is difficult to find an optimal action if a hunter learns other hunters' actions in the final stage of learning. The learning rate of learning other hunters' activity histories should be decreased in proportion to the number of episodes. If other hunters' activity histories are used at the last stage of learning, learning accuracy will reduce slightly. It does not become bad by learning only for one's history, and the learning rate at the time of updating for other hunters' activity histories should be gradually made small.

The influence of other hunters' activity histories on learning was reduced with the number of times of learning. This method (hereinafter referred to as Turned Experience CMQL (TECMQL)) is a learning approach that utilizes other hunters' activity histories in the early stage of learning and only its own history in the final stage.

The following formula defines the learning rate at learning other hunters' activity histories.

$$\alpha_{other} = \frac{\alpha}{1 + (episode / rate)} \quad (6)$$

where, $\alpha_{other}$ is a learning rate the updates the Q-value using other hunters' activity histories and *rate* is a constant that determines reduction rate of the learning rate. The learning rate at learning using other hunters' actions should be decreased according to the number of episodes. The value of parameter *rate* is determined to eliminate the effect of other hunters' actions in proportion to the number of episodes.

## V. EXPERIMENTS

In this section, the proposed method was applied to hunter games to confirm its validity.

### A. Outline of Experiments

Experiments compare learning efficiency of the following three methods.

- Proposed method: CMQL using other hunters' activity histories (referred to as Sharing Experience CMQL (SECMQL)).

- Compared method: CMQL using only each hunter's history (referred to as Own Experience CMQL (OECMQL)).

- Conventional method: CMQL that does not use activity histories.

These three methods were applied to a hunter game in a maze environment and two-prey hunter game.

### B. Experiment 1: Hunter Games in Maze Environment

The performances of the three methods mentioned above were compared in the hunter game in a maze environment. In this case, hunters learn ways of bypassing walls in the maze and leading a prey to the place where it is easy to catch using the walls. The positions of the walls do not change from the beginning of this experiment. Walls are grasped by the absolute coordinate system. In this experiment, a partial state of CMQL consists of a relative coordinate from a hunter to any other hunter, a relative coordinate from the hunter to a prey, and an absolute coordinate of the hunter itself. Actions can be learned considering the positions of walls in each partial state.

Experimental conditions were as follows:

- Size of field: 8 × 8
- Number of walls in the mazy field: 21 (cf. Fig. 7)
- Number of hunters: $n$ = 3
- Action selection strategy: ε-greedy (ε = 0.01)
- Prey's action: It escapes from hunters.
- Capture state: Four lattices in left, right, top, and bottom of a prey's position are surrounded by hunters or walls.
- Cost per one time step: 0.05
- Learning rate: α = 0.2
- Discount rate: γ = 0.8
- Maximum number of learning episodes: 300000

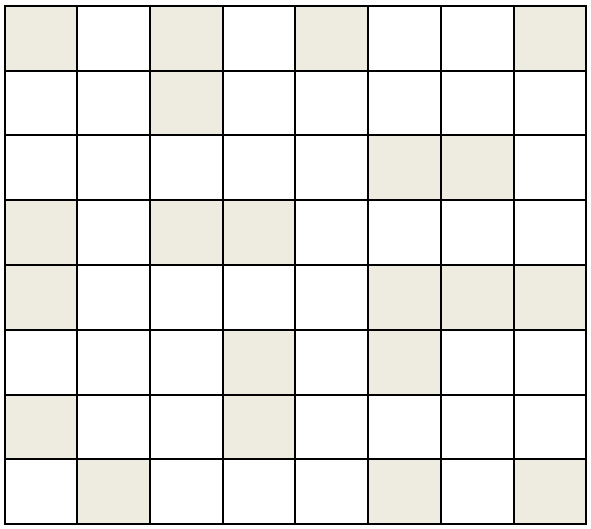

Figure 7. Maze environment.

- Reward of hunter that caught a prey directly: 5
- Reward of hunter that did not caught the prey directly: 4

In this experiment, only three hunters cannot catch a prey without making use of walls. Hunters will learn actions that guide the prey near walls and catch it using the walls. At least two or fewer hunters can catch a prey if they use walls. In this case, one hunter could guide the prey for the other two hunters to catch it. A slightly reduced reward was given to the hunter that did not catch the prey directly for its contribution to the catching.

Results are shown in Fig. 8. The horizontal axis indicates the number of episodes and the vertical axis indicates the time steps to catch a prey from an initial state. Every plot shows the average time steps to catch a prey of every 100 episodes. The fewer the time steps results in better action patterns that can be learned.

The learning of SECMQL became earlier until near episode no. 5000 than other methods, but the final learning result was bad compared with other methods. On the other hand, OECMQL could catch with fewer steps compared with CMQL. SECMQL's learning accuracy was deteriorated by learning other hunters' actions in the final stage of learning.

### C. Experiment 2: Two-Prey Hunter Games

The performances of the three methods mentioned above were compared in a hunter game that has two preys. In this game, the hunters' purpose is to catch one of the two preys. Given that the candidate actions of a hunter increases in number, learning becomes difficult compared with the problem of one prey. In this experiment, a partial state of CMQL consists of a relative coordinate from a hunter to any other hunter and two relative coordinates from the hunter to two preys. Given that the positions of both preys can be seen, actions can be learned considering the two preys.

Experimental conditions were as follows:

- Size of field: 8 × 8
- Number of hunters: $n = 3$
- Action selection strategy: ε-greedy ($\varepsilon = 0.01$)
- Prey's action: It escapes from hunters.
- Capture state: At least two hunters exist in left, right, top, and bottom of one prey.
- Cost per one time step: 0.05
- Learning rate: $\alpha = 0.2$
- Discount rate: $\gamma = 0.8$
- Maximum number of learning episodes: 300000
- Reward of hunter that caught a prey directly: 5
- Reward of hunter that did not catch the prey directly:4

In this experiment, preys observe all hunters' positions and they escape from hunters based on the hunters' coordinates. A slightly reduced reward was given to the hunter that did not catch a prey directly for its contribution to catching.

Results are shown in Fig. 9. In this experiment, the learning efficiency of SECMQL is the best in the early stages of learning. Given that action patterns that lead to catching in the early stages of learning by only one hunter are insufficient, it is useful to use other hunters' activity histories for learning.

However, OECMQL found good action strategies over 100000 episodes. Obtaining good action strategies improves the way a hunter individually learns in the final stage.

### D. Experiment 3: Hunter Games in Maze Environment after Control of Learning Rate

Performance was compared with the cases where they are with or without reducing learning rate of hunter games in a maze environment. TECMQL was added to the three methods of Experiments 1 and 2 as a compared method. Experimental conditions were the same as Experiment 1, and the *rate* of TECMQL was 500.

Results are shown in Fig. 10. In this experiment, OECMQL shows the best learning result. TECMQL also showed almost equivalent learning result to that of OECMQL, while TECMQL maintained good efficiency in the early stage of learning.

### E. Experiment 4: Two-Prey Hunter Games after Control of Learning Rate

Performance was compared with the cases where they are with or without reducing learning rate of hunter games that have two preys. The compared method was the same as Experiment 3. Experimental conditions were the same as Experiment 2, and the *rate* of TECMQL was 10000. The results are shown in Fig. 11.

In this experiment, TECMQL discovered a strategy that could catch a prey with fewer steps than other methods. From these results, it seems to be effective to assemble a rough action strategy using actions of other hunters in the early stages of learning, and then to learn the action strategy that is suitable for each hunter by individual learning.

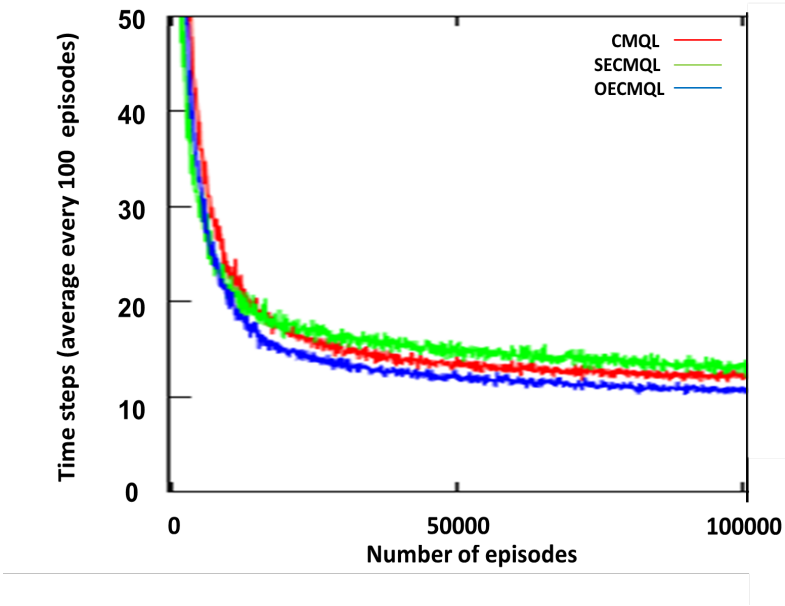


Figure 8. Results of the proposed method for maze task.

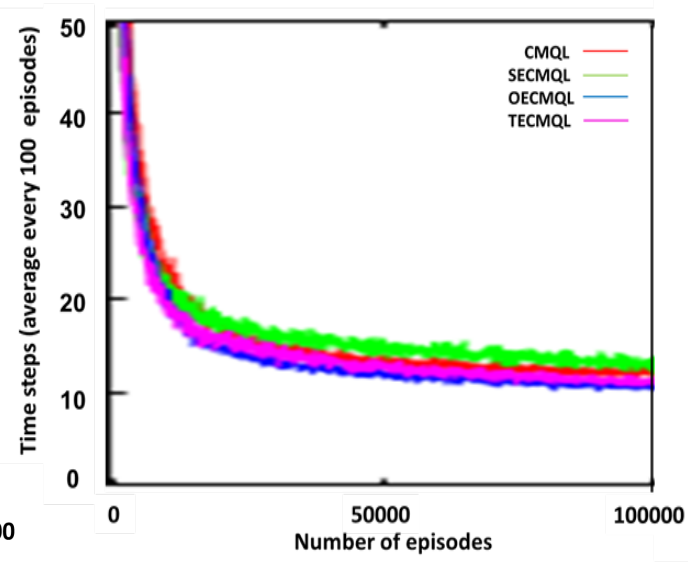


Figure 10. Results of the proposed method for maze task after control of the learning rate.

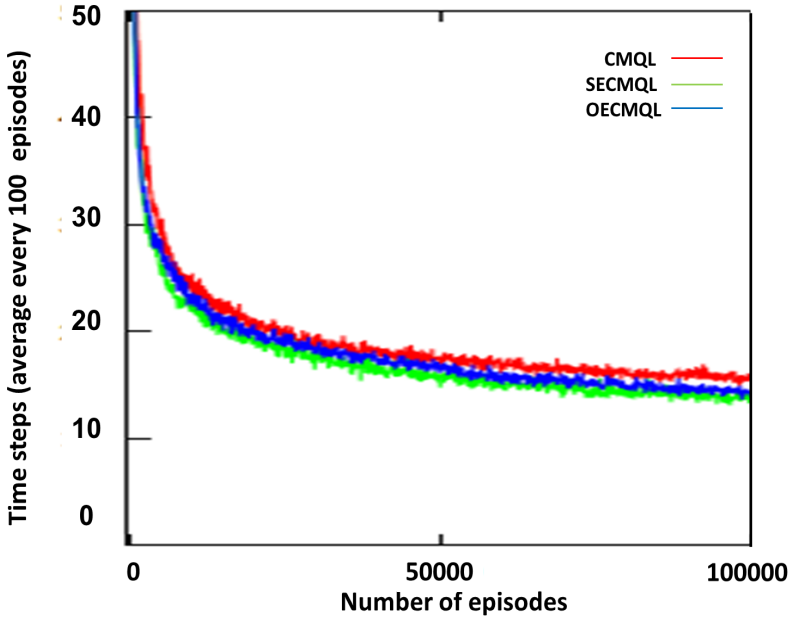


Figure 9. Results of the proposed method for two-prey game.

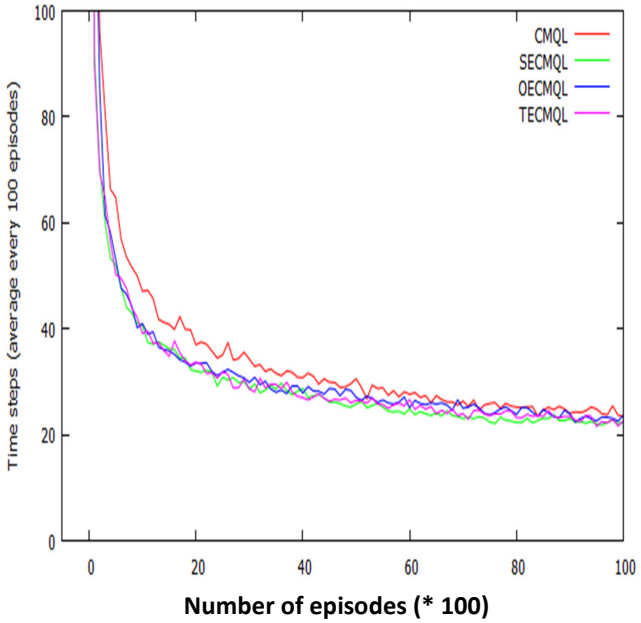


Figure 11. Results of the proposed method for two-prey game after control of the learning rate.

In addition to Experiment 3, TECMQL found actions that were easy to catch a prey rather than the conventional methods in different environments. However, it is necessary to adjust the learning rate according to the environments.

TABLE II. Convergent average steps.

| Average number of steps | Maze task game | Two-prey game |
|---|---|---|
| TECMQL | 9.6 | 10.6 |
| SECMQL | 11.5 | 12.1 |
| OECMQL | 9.4 | 11.3 |
| CMQL | 10.6 | 13.6 |

### *F. Convergent Average Number of Steps for Each Method*

Table II shows the convergent average steps for each method after each method has finished learning. TECMQL obtained the best average number of steps for the two-prey game. For the maze task game, it obtained nearly the best number of steps.

## VI. CONCLUSION

In this study, some Q-learning algorithms were applied in the hunter game of maze and two-prey environments. The composition of appropriate partial states was examined. This paper proposed a method that can learn in fewer trials by sharing activity histories among hunters. The method is based on MQL and CMQL, which are methods that prevent an explosion of the number of states. The performance of the proposed method was compared with CMQL. To solve the problem of deteriorating learning performance of the proposed method in the later stage of learning when using other hunters' activity histories, the learning rate is decreased according to the number of episodes. The proposed method can be generalized to other multiagent environment other than hunter games because it uses a general Q-learning algorithm.

At the present method, the control of learning rate is dependent on the number of episodes, but it is not controlled by the contents of learning. In future study, an index should be established to control the learning rate according to Q-value during learning. In addition, it is considered that the internal states of agents will be optimized by clustering.

## ACKNOWLEDGMENT

This work was supported in part by JSPS KAKENHI Grant Number JP16K06424.

## REFERENCES

[1] K. Matsumoto, T. Gohara, and N. Mori, "Learning method by sharing activity logs in multiagent environment," Proc. of the Tenth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2016), IARIA, October 2016, pp. 71-76, ISBN: 978-1-61208-506-7.

[2] G. Weiss, Multiagent Systems: a modern approach to distributed artificial intelligence, MIT Press, 1999.

[3] S. J. Russell and P. Norving, Artificial intelligence: a modern approach, Prentice-Hall, Englewood Cliffs, 1995.

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction, MIT Press, 1998.

[5] H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in Reinforcement Learning, Springer Berlin Heidelberg, pp. 207-251, 2012.

[6] M. Benda, V. Jagannathan, and R. Dodhiawalla, On optimal cooperation of knowledge sources, Technical Report, BCS-G 2010-28, Boeing AI Center, 1985.

[7] I. Nahum-Shani, M. Qian, D. Almirall, W. E. Pelham, B. Gnagy, G. A. Fabiano, and S. A. Murphy, "Q-learning: A data analysis method for constructing adaptive interventions," Psychological methods, vol. 17, no. 4, p. 478, 2012.

[8] S. Shamshirband, A. Patel, N. B. Anuar, M. L. M. Kiah, and A. Abraham, "Cooperative game theoretic approach using fuzzy Q-learning for detecting and preventing intrusions in wireless sensor networks," Engineering Applications of Artificial Intelligence, vol. 32, pp. 228-241, 2014.

[9] N. Ono and K. Fukumoto, "Multi-agent reinforcement learning: a modular approach," Proc. of AAAI ICMAS-96, pp.252-258, 1996.

[10] M.Tan, "Multi-agent reinforcement learning : independent vs. cooperative agents," Proc. of the 10th International Conference on Machine Learning, pp. 330-337, 1993.

[11] R. M. Kretchmar, "Parallel reinforcement learning," Proc. of the 6th World Conference on Systemics, Cybernetics, and Informatics, vol. 6, pp. 114-118, 2002.

[12] K. Hwang, W. Jiang, and Y. Chen, "Model learning and knowledge sharing for a multiagent system with Dyna-Q learning," IEEE Transactions on Cybernetics, vol. 45, no. 5, pp. 978-990, 2015.

[13] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, "FMRQ-A multiagent reinforcement learning algorithm for fully cooperative tasks," IEEE Transactions on Cybernetics, vol. 47, no. 6, pp. 1367-1379, 2017.

[14] K. Matsumoto, T. Ikimi, and N. Mori, "A switching Q-learning approach focusing on partial states," Proc. of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control (MIM 2013) IFAC, pp. 982-986, ISBN: 978-3-902823-35-9, June 2013.

[15] H. Iima and Y. Kuroe, "Swarm reinforcement learning algorithm based on exchanging information among agents," Transactions of the Society of Instrument and Control Engineers, vol. 42, no. 11, pp. 1244-1251, 2006 (in Japanese).

[16] S. Yamawaki, Y. Kuroe, and H. Iima, "Swarm reinforcement learning method for multi-agent tasks," Transactions of the Society of Instrument and Control Engineers vol. 49, no. 3, pp. 370-377, 2013 (in Japanese).

[17] T. Tateyama, S. Kawata, and Y. Shimomura, "Parallel reinforcement learning systems using exploration agents," Transactions of the Japan Society of Mechanical Engineers Series C vol. 74, no. 739, pp. 692-701, 2008 (in Japanese).

[18] Y. M. De Hauwere, P. Vrancx, and A. Nowe, "Future sparse interactions: A MARL approach," Proc. of the 9th European Workshop on Reinforcement Learning, pp. 1-3, 2011.

[19] H. Igarashi, M. Handa, S. Ishihara, and I. Sasano, "Agent control in multiagent systems – Reinforcement learning of weight parameters in particle swarm optimization," The Research Reports of Shibaura Institute of Technology, Natural Sciences and Engineering vol. 56, pp. 1-8, 2012 (in Japanese).

[20] C. J. C. H. Watkins and P. Dayan, "Technical note Q-learning," Machine Learning, vol. 8, no. 3, pp. 279-292, 1992.

[21] S. J. Bradtke and M. O. Duff, "Reinforcement learning method for continuous-time Markov decision problems," Advances in Neural Information Processing Systems, vol. 7, pp. 393-400, 1994.

[22] A. Ito and M. Kanabuchi, "Speeding up multi-agent reinforcement learning by coarse-graining of perception — hunter game as an example—," IEICE Trans. Information and

Systems D-I, vol. J84-D-I, no. 3, pp. 285-293, 2001 (in Japanese)

[23] G. Giannakopoulos and P. Themis, "Revisiting the effect of history on learning performance: The problem of the demanding lord," Knowledge and information systems, vol. 36, no. 3, pp. 653-691, 2013.

[24] I. Koychev and R. Lothian, "Tracking drifting concepts by time window optimisation," Proc. of the twenty-fifth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, pp. 46–59, 2005.

[25] I. Koychev and I. Schwab, "Adaptation to drifting user's interests," Proc. of ECML2000 Workshop: Machine Learning in New Information Age, pp. 39–45, 2000.

[26] J. Patist, "Optimal window change detection," Data Mining Workshops, pp. 557–562, 2007.