Greedy Action Selection and Pessimistic Q-value Updating in Multi-Agent Reinforcement Learning with Sparse Interaction

Toshihiro Kujirai

Supervisor: Prof. Takayoshi Yokota

July 2019

Abstract

Although multi-agent reinforcement learning (MARL) is a promising method for learning a collaborative action policy that will enable each agent to accomplish specific tasks, the state-action space increased exponentially. Coordinating Q-learning (CQ-learning) effectively reduces the state-action space by having each agent determine when it should consider the state of another agent on the basis of a comparison between the immediate rewards in a single-agent environment and those in a multi-agent environment and augment its state including the state of another agent. This thesis points out that CQ-learning has at least six issues to be improved; namely (1) how prelearning should be conducted, (2) unnecessary exploration by ϵ-greedily action selection, (3) optimistic Q-value updating, (4) which Q-values should be used if more than two agents involve in an interference, (5) a problem in a multi-agent environment must be manually converted multiple problems in a single-agent environment, and (6) it only detects a difference of immediate rewards to identify interfered states. This thesis presents four approaches to solve the issues of (1)-(4). The first approach for prelearning is to set ϵ value of ϵ-greedily action selection in a single-agent environment to 0.8 to ensure that an agent can explore all state-action combination enough. In the second approach for avoiding unnecessary exploration, when an agent is under an unaugmented state exploiting knowledge learned as a single-agent, the agent selects its action greedily. The third approach for avoiding optimistic Q-value updating is to modify an update equation based on whether an agent is still in an interference state after taking previous action. In the last approach for dealing with interference among more than two agents, one of these agents are randomly selected. This thesis calls the method GPCQ-learning when all the four approaches are applied. Evaluation using five maze games demonstrates that GPCQ-learning substantially reduces the average number of steps to the goals in comparison with CQ-learning. Then, this thesis points out that in some pursuit games GPCQ-learning fell into a deadlock because of three reasons; namely (1) a repetitive sequence of actions causes no difference of instant reward between in a single-agent environment and in a multi-agent environment,

(2) an agent selects its action greedily in an unaugmented state, and (3) the greedy action in the unaugmented state is fixed. This thesis proposes two approaches to break a deadlock caused by GPCQ-learning. The first approach is directly detecting the deadlock and augmenting the unaugmented state. The second approach is updating Q-values of unaugmented states as well as augmented states. Evaluation using seven patterns of pursuit games and five maze games demonstrates that these two approaches break the deadlock then the performance of GPCQ-learning is improved.

Acknowledgements

During the course of this work, I have been fortunate to receive assistance and advices from many individuals. I would especially like to thank my supervisor Professor Takayoshi Yokota for his continuous support, encouragement, and advices throughout this work.

I am also very grateful to the members of my thesis review committee, particularly Professor Shuhei Kimura and Associate Professor Makoto Ohki for their invaluable comments and criticism of the thesis.

I would like to express my thanks to many colleagues in Intelligent Information System department at Hitachi Ltd. who game me useful comments and time to work on this.

Finally, I would like to thank to my wife and children who have been patient while I spent lot of time on this work on weekends.

List of Publications

(1) Journal Paper

[1] T. Kujirai and T. Yokota: Greedy action selection and pessimistic Q-value updating in multi-agent reinforcement learning with sparse interaction, SICE Journal of Control, Measurement, and System Integration, Vol. 12, No. 3, pp. 76-84 (2019)

(2) Presentations in International Conferences with peer review

[1] T. Kujirai and T. Yokota: Greedy action selection and pessimistic Q-value updates in cooperative Q-learning, In Proceedings of the SICE Annual Conference, pp. 821-826 (2018).

[2] T. Kujirai and T. Yokota: Breaking a deadlock in multi-agent reinforcement learning with sparse interaction, Accepted, Pacific Rim International Conference on AI (2019)

List of Figures

List of Tables

List of Algorithms

Contents

Chapter 1 Introduction

1.1 Rational Agents and Environments

In [Russell & Norvig (2003)], an agent is defined as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." What an agent perceives from the environment is called ***observations*** and how it acts upon the environment is called ***actions*** as shown in Fig. 1.1.

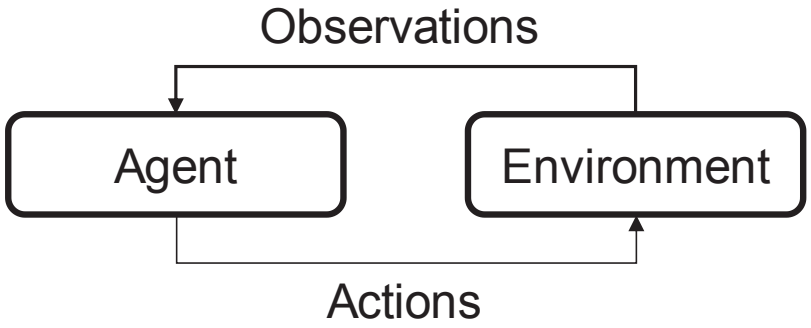


Fig. 1.1 An agent and its environment.

An agent has its ***objective***. For example, the objective of the agent illustrated in Fig. 1.2 is to maintain the pole on the car not to fall off as long as possible. The agent perceives the angle of the pole and its derivative value as an observation from the environment. The agent selects actions, Right or Left, at every time step to keep the pole upright as long as possible. The agent is called an ***rational agent***, when it has a capability to select an action that is expected to maximize its ***performance measure***, i.e. how long it keeps the pole upright in Fig. 1.2, given the evidence provided by the percept sequence and whatever built-in knowledge which it has.

The problem is how we rationalize an agent. Defining rules is an approach for it. However, it is difficult to manually define such rules to maximize its performance measure. For example, even if we define two rules, i.e. (1) select Right if $\theta>0$ and (2) select Left if $\theta<0$ for the system in Fig. 1.2, these don't rationalize the agent.

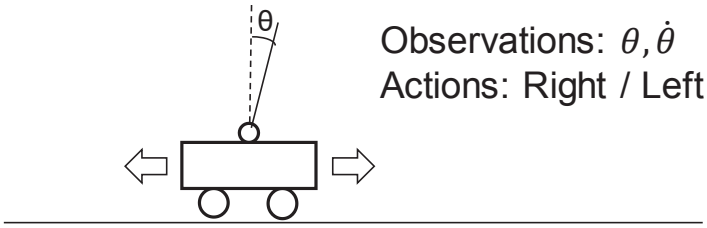


Fig. 1.2 An example of an agent system.

Another approach to rationalize an agent is ***learning from experience***. An agent does trial and error in an environment to learn how to maximize its performance measure. In this thesis, we focus on the latter approach, especially on ***reinforcement learning*** [Kaelbling et al. (1996)].

An environment is described either ***deterministic*** or ***stochastic***. In a deterministic environment, next observation is simply specified by the sequence of observations and actions, while in a stochastic environment, next observation deviates even on the same sequence of observations and actions. An environment is also described either ***stationary*** or ***unstationary***, which means that behavior of the environment changes with time. It is more difficult for agents to learn from experience in an unstationary environment. Throughout this thesis, we assume that environments are deterministic and stationary.

1.2 Multi-Agent Systems

In many situations multiple agents coexist and interact with other agents in an environment [Vlassis (2007), Bloembergen et al. (2015)]. Such a system that consists of a group of agents that can potentially interact with each other is called a ***multi-agent system*** (MAS). Soccer playing robots [Stone & Sutton (2001)] is an example of multi-agent systems.

A system consists of several sub-systems, which contains several agents. All the behavior of the system is not able to be expressed. The interaction between such the sub-systems complicates the behavior of the whole system. Therefore, an implementation of a program to control the multi-agent system is more difficult. The system controlled by such a program behaves unexpectedly when the agents are under an unconsidered

situation. Therefore the approach of learning from experience, including reinforcement learning, is supposed to be a better way to deal with MAS.

This thesis will explain differences between single-agent systems and MAS from several perspectives.

1.2.1 Shared Environment and Communication

Agents share the same environment as shown in Fig. 1.3, i.e. each agent perceives the sequence of observations from the shared environment and select actions. Even if we assume that the environment is stationary, it appears dynamic to the agents due to the other agents.

In MAS, an agent doesn't always observe the information of the other agents. If an agent can obtain all information about an environment including other agents, the agent has a full observability. Contrastly an agent that can not obtain all the information has a partial observability.

Each agent sometimes directly communicates with the other agents to obtain the information of them to collaborate. Communication has a downside due to the communication cost and computational cost.

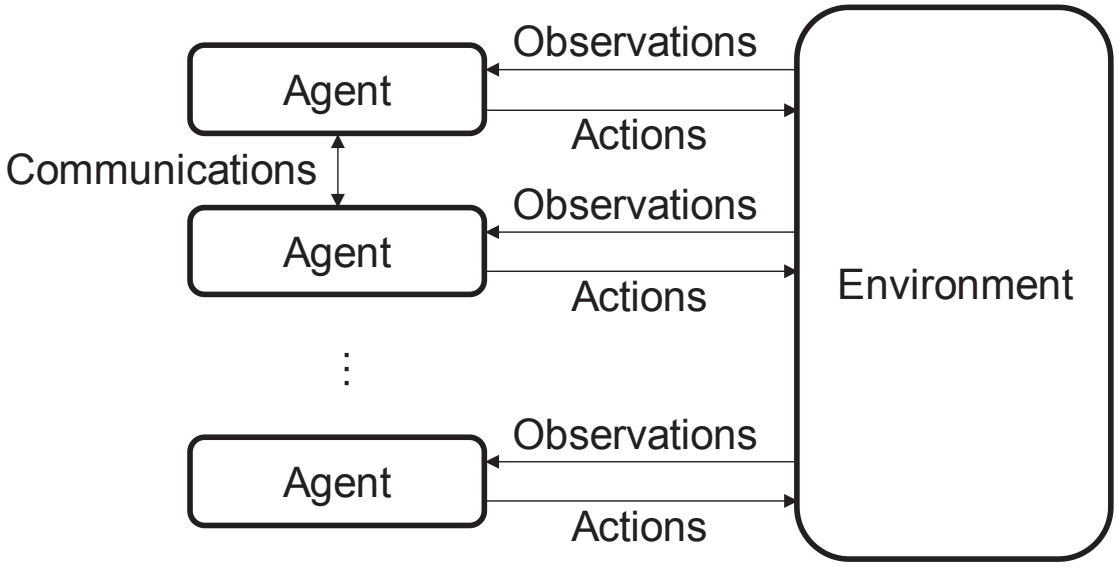


Fig. 1.3 A shared environment and communications in a multi-agent system.

1.2.2 Type of Agents

If a MAS consists of identical agents, they are called ***homogeneous***. In many cases, a MAS consists of different types of agents, i.e. a keeper, forwards, defenders, etc. Such a system is called ***heterogeneous***. This thesis deals with both MASs, namely homogeneous maze games and heterogeneous pursuit games.

1.2.3 Objectives

This thesis deals with a cooperative or team MAS. Each agent of this type has the same objectives. For example, a robot playing soccer in a team has the same objectives; to score a goal and avoid other team's goal while robots in another team has opposite objectives. If robots in a MAS has the same objectives, the system is called a cooperative or team MAS.

1.3 Sparse Interaction in MASs

In MASs, each agent decides its optimal action based on states and actions of other agents. However, in many cases they decide their optimal action independently most of time due to sparse interactions between agents.

Figure 1.3 shows an example of sparse interaction in a MAS. The objective of each agent is to move the heavy container to a specific location. Both agents must collaborate to move the container because it is too heavy for a single agent. To achieve the objective, both agents firstly approach the container. In the process of approaching it, each agent does not consider the state of another agent. Once each agent comes near to the container, each agent considers the state of another agent because (i) there are unneglectable risk of collisions between agents and (ii) each agent locates on the same side of the container, and (iii) each agent locates next to another agent.

The assumption of sparse interaction reduces the combination of states and actions of both agents because each agent independently decides its action most of time.

This thesis focuses on sparse interaction in MASs.

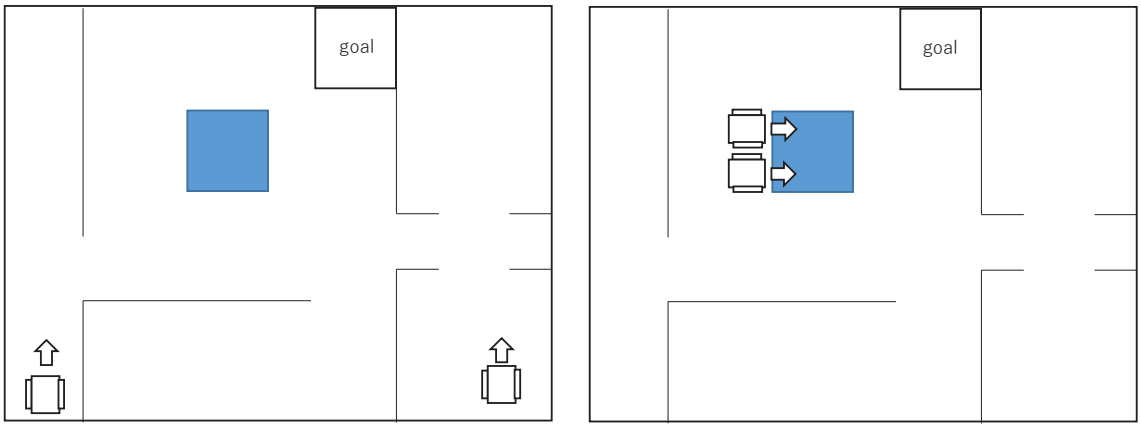


Fig. 1.4 An example of the sparse interaction

1.4 Contributions of the thesis

- This thesis points out that CQ-learning, which is one of reinforcement learning method for Dec-SIMDP, has at least four issues to be improved; namely how prelearning should be conducted, unnecessary exploration by ϵ-greedily action selection, optimistic Q-value updating, and which Q-values should be used if more than two agents involve in an interference.

- This thesis presents four approaches to solve the issues. The first approach for prelearning is to set ϵ value of ϵ-greedily action selection in a single-agent environment to 0.8 to ensure that an agent can explore all state-action combination enough. The second approach for avoiding unnecessary exploration is making an agent select its action greedily if it is in an unaugmented state exploiting knowledge learned in a single-agent environment. The third approach for avoiding optimistic Q-value updating is to change Q-value updating equation based on whether an agent is still in an interference state after taking previous action. The last approach for dealing with interference among more than two agents is randomly selecting one agent from agents that are in the interference state.

- Evaluation using five maze games demonstrates that if both greedy action selection and changing Q-value updating equation based on whether an agent is still in an

interference state after taking previous action are applied, we call the learning method GPCQ-learning, the performance of CQ-learning is improved substantially.

- The thesis points out that in some pursuit games GPCQ-learning fell into a deadlock due to greedy action selection at an unaugmented state and failing to detect the deadlock because there was no difference of instant reward between in a single-agent environment and in a multi-agent environment.

- This thesis proposes two approaches to break a deadlock caused by GPCQ-learning. The first approach is directly detecting the deadlock and augmenting the unaugmented state. The second approach is updating Q-values of unaugmented states as well as augmented states.

- Evaluation using seven patterns of pursuit games and five maze games demonstrates that the two proposed approaches improved the performance of GPCQ-learning by breaking a deadlock.

1.5 Outline of the thesis

The remainder of the thesis consists of as follows.

Chapter 2 describes a background of reinforcement learning for a single-agent environment. This thesis starts with defining an MDP (Markov Decision Process) that is a basic problem statement of long-term optimization problems and introducing value functions. This thesis explains three basic methods to efficiently solve MDPs; Dynamic Programming, Monte Carlo methods, and Temporal-Difference (TD) learning. This thesis also describes the detailed algorithm of Q-learning that is the most well-known TD learning method.

In Chapter 3, some characteristics of a multi-agent system are described. This thesis

explains some extensions of MDP to deal with some class of problem in a multi-agent system. Maze games are introduced as an example of a multi-agent system. This thesis presents there is a problem of explosion of state-action space in a multi-agent system and demonstrated how the explosion affects the performance of reinforcement learning methods. This thesis introduces a concept of sparse interaction, which dramatically reduces state-action space, and formulated such a problem as Dec-SIMDP (Decentralized Sparse Interaction MDP). This thesis finally introduces several reinforcement learning methods for Dec-SIMDP including Coordinating Q-learning (CQ-learning), which proposed methods in this thesis are based on. CQ-learning effectively reduces the state-action space by having each agent determine when it should consider the states of other agents based on a comparison between the immediate rewards in a single-agent environment and those in a multi-agent environment.

In Chapter 4, this thesis points out that CQ-learning has at least four issues to be improved; namely (1) how prelearning should be conducted, (2) unnecessary exploration by ϵ-greedily action selection, (3) optimistic Q-value updating, and (4) which Q-values should be used if more than two agents involve in an interference.

This thesis presents four approaches to solve the issues. The first approach for prelearning is to set ϵ value of ϵ-greedily action selection in a single-agent environment to 0.8 to ensure that an agent can explore all state-action combination enough. The second approach for avoiding unnecessary exploration is making an agent select its action greedily if it is in an unaugmented state exploiting knowledge learned in a single-agent environment. The third approach for avoiding optimistic Q-value updating is to change Q-value updating equation based on whether an agent is still in an interference state after taking previous action. The last approach for dealing with interference among more than two agents is randomly selecting one agent from agents that are in the interference state.

Evaluation using five maze games demonstrates that if both greedy action selection and changing Q-value updating equation based on whether an agent is still in an interference state after taking previous action are applied, this thesis calls the learning

method GPCQ-learning, the performance of CQ-learning is improved substantially.

In Chapter 5, this thesis points out that in some pursuit games GPCQ-learning fell into a deadlock due to greedy action selection at an unaugmented state and failing to detect the deadlock because there is no difference of instant reward between in a single-agent environment and in a multi-agent environment.

This thesis proposes two approaches to break a deadlock caused by GPCQ-learning. The first approach is directly detecting the deadlock and augmenting the unaugmented state. The second approach is updating Q-values of unaugmented states as well as augmented states.

Evaluation using seven patterns of pursuit games and five maze games demonstrates that the two proposed approaches improved the performance of GPCQ-learning by breaking a deadlock.

Chapter 6 concludes the thesis by restating our contributions and describing some future work in this domain.

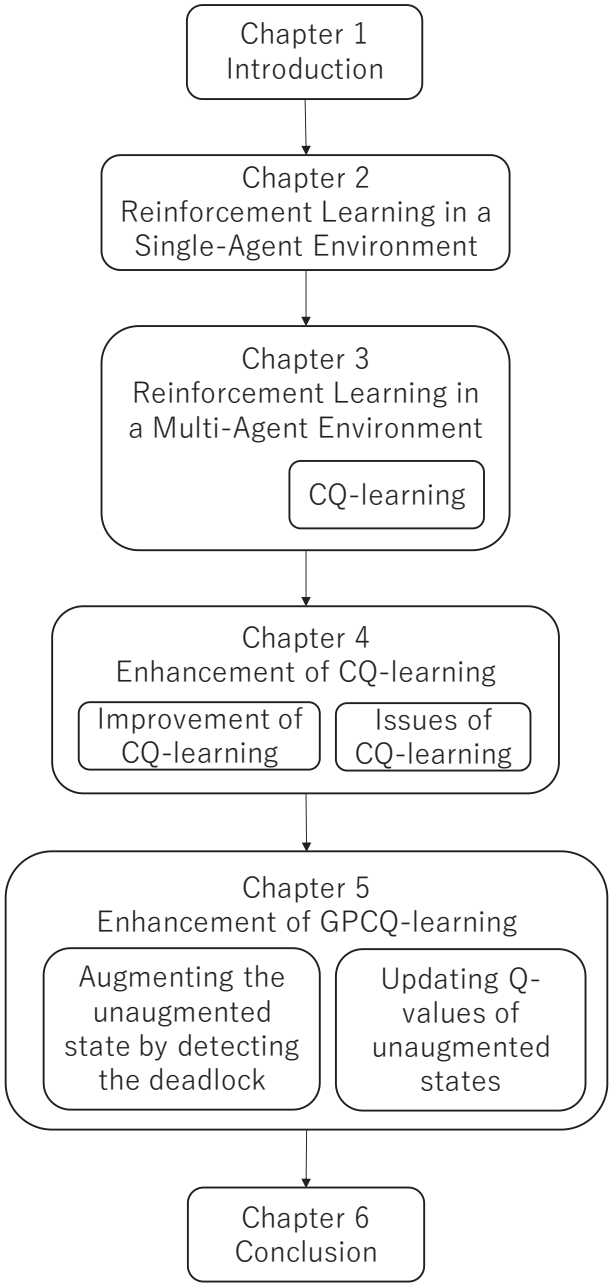


Fig. 1.5 Outline of this thesis.

Chapter2 Reinforcement Learning in a Single-Agent Environment

An agent makes decisions to achieve its objective. In this chapter, we address how the decision-making problem is mathematically defined and how the problem is solved in a single-agent environment.

2.1 Markov Decision Process (MDP)

Optimal control is a technical term that describes problems in which a design of a controller is made to minimize or maximize a criterion of a dynamic system through a time sequential process. Bellman [Bellman (1957)] formulated a method of the sequential decision problems for an ***optimal control***, which is known as ***Dynamic Programming***. At each step of a decision cycle, an agent observes a state of a system and decides its action. The decision influences the state of the system at the next step. An immediate reward is gained based on the state and the decision. The purpose of the optimal control is to maximize the expected cumulative reward through the decision process.

A ***Markov Decision Process*** (***MDP***) is one of mathematical formulations of such a decision process. A MDP can be defined as a tuple (S, A, T, R), where S stands for a finite set of states of an environment, A stands for a finite set of actions, $T(= p(s'|s, a))$ and $R(= r(s, a, s'))$ stand for the transition probability matrix and immediate reward matrix for the combinations of state $s \in S$, action $a \in A$, and next state $s' \in S$.

An MDP has the ***Markov Property***, which means that its transition probability matrix is defined only by the current state and action, and the previous states and actions do not affect the transition.

$$T\big(s(t+1)\big|s(t), a(t), \dots, s(0), a(0)\big) = T(s(t+1)|s(t), a(t))$$

A cumulative reward from t is defined as follows.

$$R_t = \sum_{k=0}^{\infty} r_{t+k+1}$$

In a ***discounted MDP***, a future reward is discounted by $\gamma \in [0,1]$ for every step, which makes an agent tend to weight on an immediate reward. A discounted cumulative reward from t is defines as follows. When $\gamma = 1$, the discounted MDP is equivalent to an MDP.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

A ***policy*** is described as $\pi(= p(a|s))$, which is mapping $S \times A \rightarrow [0,1]$ for all $s \in S$. The goal of an agent is to learn an optimal policy π^*, i.e., one that maximizes the expected cumulative reward through the decision process.

There are at least two approaches to solve a decision-making problem based on an MDP. The first one is to directly estimate the optimal policy, while the another is to estimate ***value functions***.

A ***state-value function*** $V^{\pi}(s)$ is described as the expected discounted cumulative reward if the agent decides its actions based on the policy π after it visits state s, and formally formulated as follows.

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\}$$

An ***action-value function*** $Q^{\pi}(s,a)$ is described as the expected discounted cumulative reward if the agent decides its actions based on the policy π after it decides action a at state s, and formally formulated as follows.

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\}$$

The Bellman equation represents a relationship between the value of a state and the value of its successor states. The Bellman equation for $V^{\pi}(s)$ is described as following.

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} p(s'|s,a)\{r(s,a,s') + \gamma V^{\pi}(s')\}$$

To solve the decision-making problem is to find a policy that maximizes value functions. For any MDP, there is at least one ***optimal policy***, such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ and $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for every π, every s, and every a. $V^{\pi^*}(s)$ is called as an ***optimal state-value function*** and noted as $V^*(s)$ for short. $Q^{\pi^*}(s,a)$ is called as an ***optimal action-value function*** and noted as $Q^*(s,a)$ for short.

The optimal value functions can be estimated by learning from experience. If we can obtain enough amount of experiences to calculate expected values of each reward of each combination of state and action, $V^*(s)$ and $Q^*(s,a)$ can be estimated. However, it is difficult to obtain enough amount of experiences due to exponential increase in the combination of a state and an action.

Bellman optimality equation gives us a method to estimate the optimal value functions iteratively and efficiently from experiences. Bellman optimality equation for an optimal state-value function $V^*(s)$ is described as

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} p(s'|s,a) \{r(s,a,s') + \gamma V^*(s')\},$$

while Bellman optimality equation for an optimal action-value function $Q^*(s,a)$ is described as

$$Q^*(s,a) = \sum_{s' \in S} p(s'|s,a) \left\{ r(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a') \right\}.$$

Once a $Q^*(s,a)$ is obtained, an optimal action policy can be described as

$$\pi^* = \operatorname*{argmax}_{a \in A} Q^*(s,a).$$

2.2 Solving Methods of MDPs

In this section, we explain three basic methods to solve MDPs, i.e., Dynamic Programming, Monte Carlo methods, and Temporal-Difference learning (TD learning). We also represent detailed algorithm of Q-learning, which is a typical method of TD learning, because all our proposed methods in this thesis are based on it.

2.2.1 Dynamic programming

If an agent knows a complete model of an MDP, optimal policies can be obtained using dynamic programming (DP). The basic idea of DP is taking advantage of value functions to improve efficiency of the search for good policies. If the complete model of the MDP is known, optimal policies can be obtained by solving following linear equation which has $|S|$ unknowns.

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\}$$
$$= \sum_{a}\pi(a|s)\sum_{s'}p(s'|s,a)\{r(s,a,s') + \gamma V^{\pi}(s')\}$$

Instead of directly solving the equation, DP takes advantage of iterative solutions, which are more suitable to obtain the optimal policies. Each successive approximation of value function under a policy can be calculated by using a following update rule. This algorithm is called ***iterative policy evaluation***.

$$V_{k+1}(s) = E_{\pi}\{R_{t+1} + \gamma V_k(s_{t+1}|s_t = s)\}$$
$$= \sum_{a}\pi(a|s)\sum_{s'}p(s'|s,a)\{r(s,a,s') + \gamma V_k(s')\}$$

To find better policies, the policy evaluated using iterative policy evaluation algorithm should be changed. If a value of action-value function $Q^{\pi}(s,a)$ for action a_i is higher than $V^{\pi}(s)$, it is better for the agent to select action a_i instead of an action selected based on π. The policy π' that selects action a_i and follows π after that is better than π. A strictly improved policy π' can be obtained using following equation. This algorithm is called ***policy improvement***.

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s\prime}p(s'|s,a)\{r(s,a,s') + \gamma V^{\pi}(s')\}$$

Once the improved policy $\pi'(s)$ is obtained, the state-value function can be estimated using iterative policy evaluation algorithm. The policy can be monotonically improved by repeating iterative policy evaluations and policy improvements as shown in Fig. 2.1 where $\overset{\mathrm{E}}{\longrightarrow}$ stands for an iterative policy evaluation, and $\overset{\mathrm{I}}{\longrightarrow}$ stands for a policy improvement. This iteration must converge to an optimal policy because each iteration

strictly improves the previous policy and a finite MDP has only a finite number of policies. This algorithm is called ***policy iteration***.

$$\pi_0(s) \xrightarrow{\text{E}} V^{\pi_0}(s) \xrightarrow{\text{I}} \pi_1(s) \xrightarrow{\text{E}} V^{\pi_1}(s) \xrightarrow{\text{I}} \cdots \xrightarrow{\text{I}} \pi_*(s) \xrightarrow{\text{E}} V^{\pi_*}(s)$$

Fig. 2.1 A policy iteration.

2.2.2 Monte Carlo methods

Monte Carlo (MC) methods learn only from experience while DP needs complete knowledge of an MDP. It must be noted that simulation can be used for the experience as well as real experience.

The idea of obtaining an optimal policy by repeating value evaluation on a policy and improvement of it is the same as DP.

MC methods collect a sequence of states, actions, and rewards in an episode. Because a state-value of $V^{\pi}(s)$ is defined as a discounted cumlative reward after visiting s, it can be estimated if enough amount of the sequences from episodes can be collected. In the episodes, an agent visits a state multiple time. There are two types of estimation of the state-value; the first is considering only the first visit (***first-visit MC***) and the second is considering every visit (***every-visit MC***). Both estimations converge to the same value if the episodes are infinite. The estimation can be independently conducted for each state. If a subset of the states is important for obtaining an optimal policy, MC methods can focus on them.

If complete knowledge of an MDP is available, evaluation of the state-value is sufficient to determine a policy, as explained the last subsection. MC methods need to evaluate action-values to determine a policy. The algorithm of estimating the action-values is basically the same as the one of estimation the state-values. The problem is that most of combination of a state and an action does not appear in the episodes if it follows the current policy, resulting in short of clue to improve the policy.

Improvement of the policy is achieved to select a greedy action based on the evaluated Q-values. The greedy policy makes the problem described above more serious.

There are two approaches to solve the problem, which are called ***on-policy*** methods and ***off-policy*** methods respectively.

In on-policy methods, an ***ϵ-soft policy***, meaning that $\pi(a|s) > 0$ for $\forall s \in S$ and $\forall a \in A(s)$, is used to ensure all state-action pairs appear in episodes. A typical ϵ-soft policy is an ϵ-greedy policy where a greedy action is selected with probability 1- $\epsilon+\frac{\epsilon}{|A(s)|}$ and one of other actions is selected with probalibity $\frac{\epsilon}{|A(s)|}$.

In on-policy methods, experience is collected selecting actions following a policy. In off-policy methods, a ***behavior policy*** μ is introduced to determine actions while a target of optimization is called a ***target policy***. It enables us to use a deterministic policy as a target policy while using an ϵ-soft policy for a behavior policy to ensure all state-action pairs appear in episodes.

2.2.3 Temporal-Difference Learning

Temporal-difference (TD) learning [Sutton (1984, 1988)] is a combination of MC ideas and DP ideas. Like MC methods, TD methods can learn directly from experience without complete knowledge of an MDP. Like DP, they update estimations based in part on other learned estimations.

Both MC methods and TD learning use experience to evaluate value functions. Instead of waiting until the end of the episode to estimate a discounted cumulative reward in MC, TD learning continuously updates its value functions during each episode. The simplest TD learning method, known as TD(0), updates its state-values using following rule,

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)].$$

TD learning bases its update in part on an existing estimation like DP. Like MC, there are two approaches for TD learning; on-policy methods and off-policy methods. We explain a typical method for each approach, i.e. ***Sarsa*** and ***Q-learning***.

In Sarsa [Rummery & Niranjan (1994)], instead of considering transitions from state to state, and learning the values of states, it considers transitions from a state-action pair

to a state-action pair, and learn the value of state-action pairs as shown in Fig. 2.2.

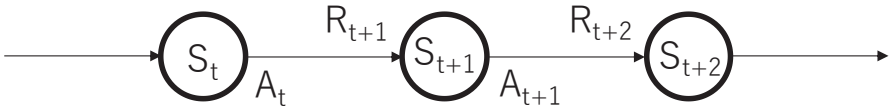


Fig. 2.2 A sequence of $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

Sarsa updates the action values based on following rule,

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - \mathrm{Q}(S_t, A_t)].$$

This update is done after every transition from a nonterminal state S_t. If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero. The name of Sarsa comes from that the update rule requires the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

Q-learning [Watkins (1988), Watkins & Dayan (1992)] is the most well-known off-policy TD learning algorithm. Its simplest form, one-step Q-learning, updates Q-values using following rule,

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - \mathrm{Q}(S_t, A_t)\right].$$

Without using A_{t+1} like in Sarsa, Q-learning can estimate an optimal Q-values independently from the policy being followed. If all state-action pairs are visited infinitely often and a suitable evolution for the learning rate is chosen, it is proven that $\mathrm{Q}(S_t, A_t)$ will converge to Q^* [Tsitsiklis (1994)]. The algorithm of Q-learning is described in Algorithm 2.1.

Algorithm 2.1: Q-learning algorithm

Initialize $\mathrm{Q}(\mathrm{s}, a), \forall s \in A(s)$, arbitrarily, and $\mathrm{Q}(\mathrm{terminal-state}, \cdot) = 0$
for each episode **do**
  Initialize S
  for each step of episode **do**
    Choose A from S using policy derived from Q (e.g., ϵ-greedily)
    Take action A, observe R, S'
    $\mathrm{Q}(\mathrm{S}, A) \leftarrow \mathrm{Q}(\mathrm{S}, A) + \alpha \left[R + \gamma \max_a Q(S', a) - \mathrm{Q}(\mathrm{S}, A)\right]$
    $\mathrm{S} \leftarrow S'$
  end for
end for

2.3 Summary

This chapter introduces a concept of MDP to a represent sequential decision-making problem and defined that solving an MDP is determing an optimal policy π^*, i.e., one that maximizes the expected cumulative reward through the decision process. Then, the state-value function $V^{\pi}(s)$ and the action-value function $Q^{\pi}(s,a)$ are defined, on which a lot of methods to solve an MDP are based. Three basic methods for solving a MDP, Dynamic Programming, Monte Carlo methods, and Temporal-Difference (TD) learning respectively, are described. Especially, we must note that all the proposed methods in this thesis are based on Q-learning, which is the most well-known TD learning method.

Chapter 3 Reinforcement Learning in a Multi-Agent Environment

3.1 Extended MDP for Multi-Agent Environment

In an MDP, it is assumed that there is a single agent in an environment and it can observe all the states of the environment (***full observability***) and the environment is stationary; i.e. states S, transition matrix T, and rewards R are fixed.

In mulit-agent environments, more than one agent exist in an environment and they obtain a part of the states of the environment, because an agent does not know about information of other agents. From a perspective of an agent, the environment looks changing because of the mere exisitance of other agents. In addition to these, differences of rewards each agent gains affect the optimal policy of each agent. For example, if all agents share same rewards, it is optimal for them to collaborate with each other to maximize cumulative rewards. Such a multi-agent system is called a ***fully cooperative game***. If a sum of rewards each agent gains is always zero in a multi-agent system, it is called a ***zero-sum game*** or a ***fully competitive game***. If each agent gains different rewards and the sum of the rewards is not zero in a multi-agent system, it is called a ***mixed competitive game***.

As shown in Table 1, an MDP can be extended for multi-agent environments in at least four ways. A natural extension is multi-agent MDP (MMDP), in which agents share all states of an environment (full observability) and rewards from the environment (fully cooperative). The agents share all their states and actions and obtain the same rewards from the environment as a result of their joint actions [Boutilier (1996)]. Another extension is decentralized MDP (Dec-MDP), in which each agent can observe only its own states, and the agents obtain the same rewards [Bernstein et al. (2002), Melo (2011)]. If they can know the complete state of the environment by sharing their observations, the agents are said to have full joint observability. These two extensions are called fully

cooperative games because the agents obtain the same rewards. In contrast, in a Markov game (MG) and a decentralized Markov game (Dec-MG), each agent has an independent reward function. This results in a competitive situation [Aras (2004)].

Table 3.1 Extended multi-agent systems.

	Full observability	Full joint observability
Shared rewards	MMDP	Dec-MDP
Independent rewards	MG	Dec-MG

3.2 Maze Games

Figure 3.1 shows five examples of maze games in which each agent i tries to find an optimal path from start position S_i to goal G_i, which are used for evaluation later. The game finishes when all the agents in the maze reach their goals. Because every movement costs a penalty , i.e. a minus reward, for each agent, they tend to learn how to minimize the steps to their goals. However, they collide if each one simply takes the shortest path, which results in a more penalty than a penalty of a movement. All the maze games were used by Hauwere et al. [Hauwere (2011)].

In this thesis, maze games are designed as a fully cooperative game because each agent gains the same reward, i.e. penalties for a movement (-1) and a collision (-10), and a reward for a goal (+0).

In the TunnelToGoal and TunnelToGoal3 games, each agent has the same goal and starts from a different position. They collide if the simply take the shortest path. At least one of them needs to wait while the others proceed. However, if the all wait, more steps will be needed to reach the goal. The optimal solution, i.e., one that minimizes the number of steps it takes for both agents to reach their goals, is for one agent takes the shortest path while another agent waits for the first agent to proceed for both games. In addition, the last agent needs to wait for the first two agents to proceed in the TunnelToGoal3 game.

In the ISR, CIT, and CMU games, the goal for each agent is the start position of the

other agent. Simply waiting will not prevent a collision. The optimal solution is for the agent taking a detour to do so immediately before a collision would occur. However, finding this solution requires extensive exploration of potential detours because there are a number of unsuitable detour routes.

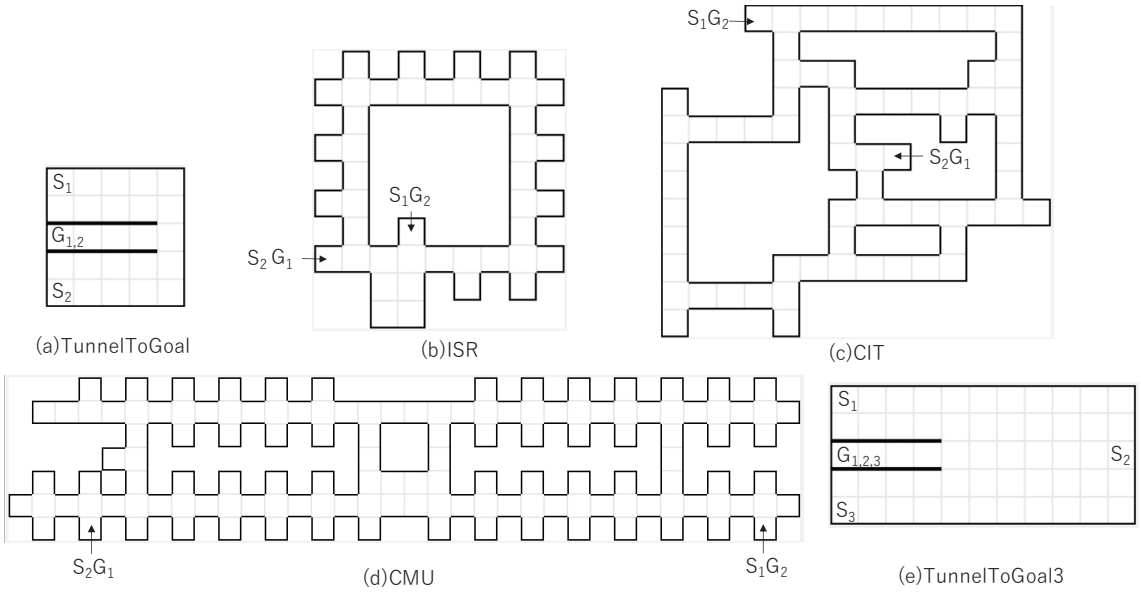


Fig. 3.1 Five examples of maze games.

3.3 Explosion of State-Action Space

If each agent shares the same reward for a task, i.e., a fully cooperative game, independent learners sometimes learn a collaborative action policy without considering the states and actions of other agents because a random exploration strategy enables them to learn collaborative actions coincidently [Lauer & Riedmiller (2000), Sen et al. (1994)]. While joint-action learners may perform better because they take information about other agents (sensation, episodes, learned policies, etc.) into account, they suffer an exponential increase in the state-action space for learning, thereby reducing the learning speed and increasing the cost of communication and the cost of estimating information about other agents [Tan (1993)].

Figure 3.2 shows the average number of steps for completing a game for every 100 episodes using three different existing methods for the five games. These methods are

straightforward extensions of Q-learning for a multi-agent environment. The first method is ***independent-learning***, which is Q-learning itself. Each agent acquires its own action policy without having any information about the other agents. The second one is ***joint-state learning*** (JSQ-learning), in which each agent always knows the states of the other agents and decides its actions independently based on its own policy. The third is ***joint-state-action learning*** (JSAQ-learning), in which one super-agent observes all the states and decides a combination of actions, i.e., joint-actions, for all the agents. So JSAQ-learning learns a joint-action policy instead of a action policy [Claus & Boutilier (1998)].

As shown in Fig. 3.2, even the agents trained using independent-learning learned how to avoid collisions and reach their goals unimpeded. In the TunnelToGoal, ISR, and CIT games, which have a small state-action space, the action policies of the agents trained using independent-learning converged the fastest although the average number of steps to the goal was the highest because these agents did not explicitly take information about other agents into account. In the CMU and TunnelToGoal3 games, which have a larger state-action space, the agents trained using independent-learning had superior performance because 10,000 episodes are not sufficient for the other methods to find the optimal policy in the large state-action space.

Policy convergence was slower for the agents trained using JSQ-learning, and the average number of steps to the goal was less than that for the agents trained using independent-learning in the TunnelToGoal, ISR, and CIT games because these agents took into account the states of the other agents when they selected their actions. In the CMU and TunnelToGoal3 games, the convergence of their policies was slower than that of the agents trained using independent-learning because they have a larger state-action space: $43 \times 43 \times 4 = 7396$ in the ISR game and $133 \times 133 \times 4 = 70{,}756$ in the CMU game, for example.

The agents trained using JSAQ-learning learned the better policy than the other two types of agents in the TunnelToGoal, ISR, and CIT games because the state-action space was small enough for each agent to learn its joint-action policy. They had difficulty learning the optimal policy in the CMU and Tunnel- ToGoal3 games because the state-

action space was too large (133 × 133 × 4 × 4 = 283,024 in the CMU game) for the agents to sufficiently explore all the state-action pairs in the limited number of episodes (10,000 in this case).

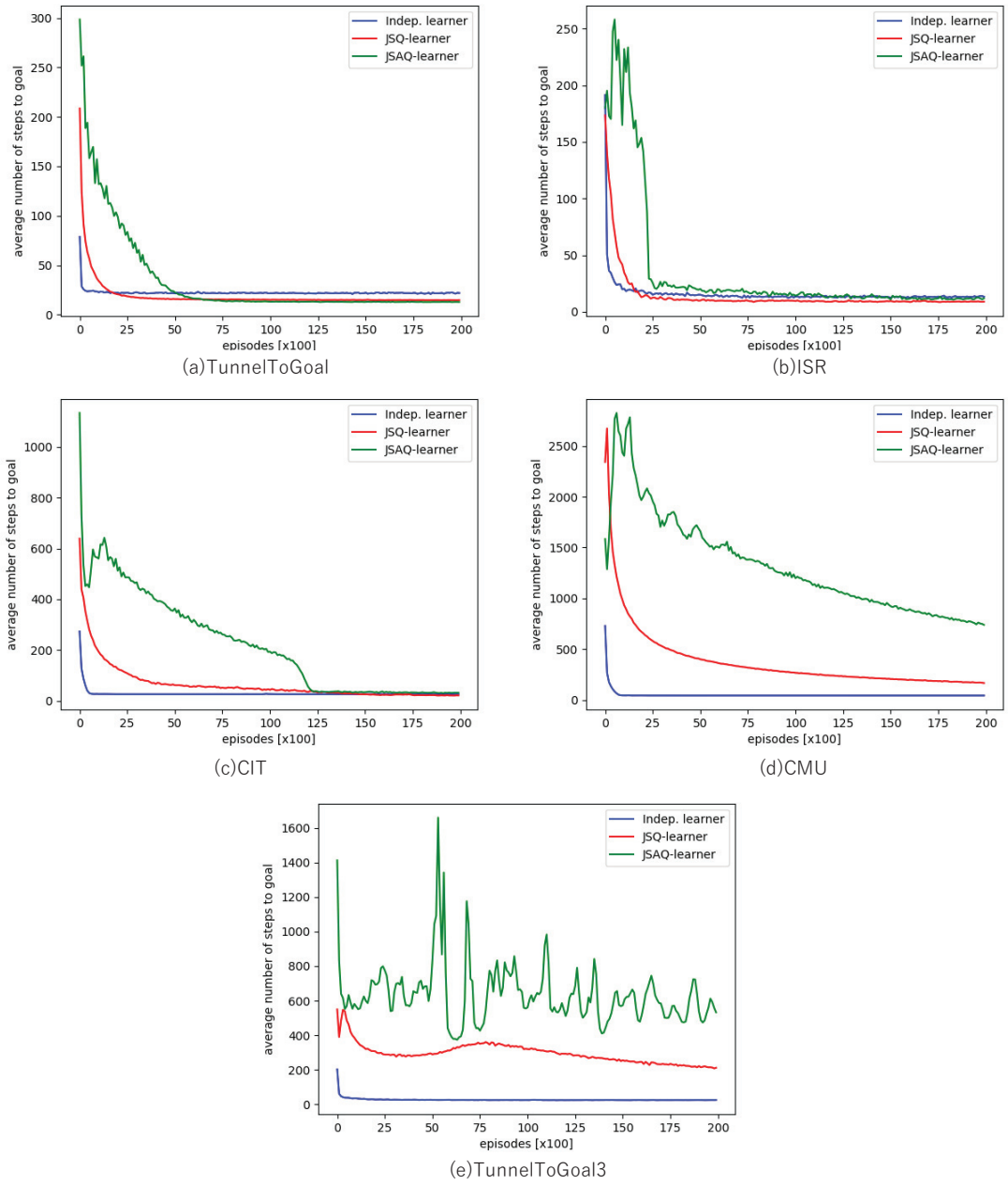


Fig. 3.2 Learning curves of three existing methods for five maze games.

3.4 Dec-SIMDP

In many cases it is not necessary for agents to consider states and actions of other agents all time and they can decide their optimal action independently most of time due to sparse interactions between agents [Guestrin et al. (2002)b].

In Fig. 3.3, each agent can independently select its action while it is in a room. They must coordinate only when they are near the narrow corridor.

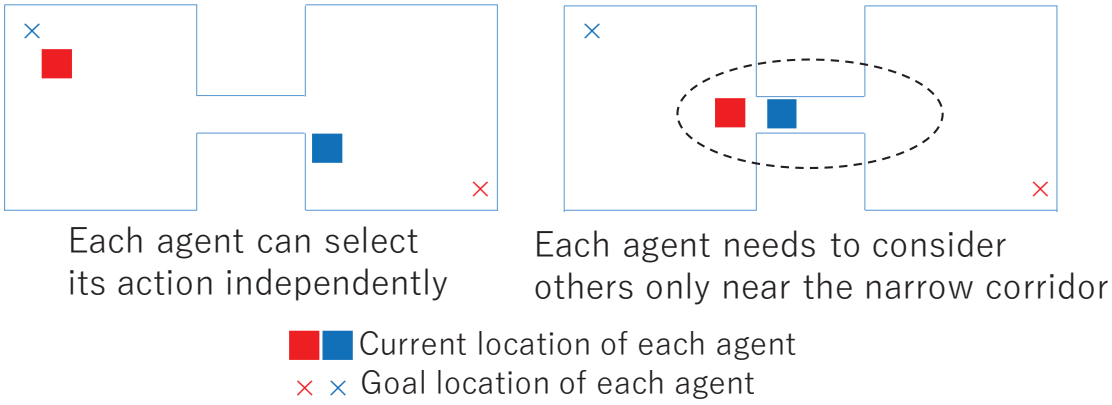


Fig. 3.3 An example of sparse interaction in a maze game.

Melo and Veloso proposed a framework of Dec-SIMDP (Decentralized Sparse Interaction MDP) that is a special case of Dec-MDP. In Dec-SIMDP, each agent rarely interferes with other agents, and it must consider a state of other agents only in limited situations [Melo & Veloso (2010, 2011)].

A Dec-MDP $\mathcal{M} = (N, (S_k), (A_k), (Z_k), P, (O_k), r, \gamma)$ is called having sparse interaction if all agents are independent except in a set of M interaction areas, $\{S_1^I, \cdots, S_M^I\}$, with $S_i^I \subset S_{K_i}$ for some set of agents K_i, and such that $|S_i^I| \ll |S_{K_i}|$. Such a Dec-MDP is referred as a decentralized sparse interaction MDP (Dec-SIMDP) and is represented as a tuple

$$\Gamma = (\{\mathcal{M}_k, k = 1, \cdots, N\}, \{(S_i^I, \mathcal{M}_i^I), i = 1, \cdots, M\}),$$

where

- each $\mathcal{M}_k$ is an MDP $\mathcal{M}_k = (S_0 \times S_k, A_k, P_k, r_k, \gamma)$ that individually models agent k in the absence of other agents, where r_k is the component of the joint reward function associated with agent k in the decomposition as following.

$$r(s,a) = \sum_{k=1}^{N} r_k(\overline{s_k}, a_k)$$

where $s = (s_0, \cdots, s_N)$, $a = (a_0, \cdots, a_N)$, and $\overline{s_k}$ is a local state $s_0 \times s_k$.

- each $\mathcal{M}_i^I$ is an MMDP that captures a local interaction between K_i agents in the states in S_i^I and is given by $\mathcal{M}_i^I = (K_i, S_{K_i}, (A_k), P_i^I, r_i^I, \gamma)$, with $S_i^I \subset S_{K_i}$ where K_i denotes a subset of the N agents in a MDP $\mathcal{M}_i$.

Each MMDP $\mathcal{M}_i^I$ describes the interaction between a subset K_i of the N agents, and the corresponding state space S_{K_i} is a superset of an interaction area as defined above.

3.5 Reinforcement Learning in Sparse Interaction

Under the assumption of sparse interaction, several reinforcement learning methods have been proposed. There are two perspectives how to exploit the assumption of sparse interaction, namely when information of other agents should be considered and how the information should be used. From the first perspective there are several methods proposed in which when information of other agents should be considered is explicitly predefined by a designer while other methods try to find it. In sub-section 3.5.1, we introduce several methods using the predefined approach. In subsections 3.5.2-3.5.5, we describe four learning methods that find when information of other agents should be considered by itself: Utile Coordination [Kok et al. (2006)], Learning of Coordination [Melo & Veloso (2009)], Entropy based approach [Arai & Xu (2016)], and CQ-learning [Hauwere (2010, 2011)]. We specially describe CQ-learning in detail because our proposal is based on CQ-learning.

3.5.1 Predefined sparse interaction

Kok and Vlassis proposed Sparse Tabular Multiagent Q-learning method [Kok & Vlassis (2004)b] in which two different type of Q-values, i.e., $Q(s_i, a_i)$ for an

uncoordinated state and $Q(\mathbf{s}, \mathbf{a})$ for a coordinated state. In an uncoordinated state, agent i selects its action based on $Q(s_i, a_i)$ independently. In a coordinated state, agents select a coordinated action based on $Q(\mathbf{s}, \mathbf{a})$. A set of coordinated states is defined in a problem specific manner. For example, they use two explicitly defined rules to distinguish a coordinated state from an uncoordinated state.

Coordination graphs are another approach for representing sparse interaction between agents. Several methods have been proposed to represent dependencies that exist between specific agents [Guestrin et al. (2002), Kok & Vlassis (2004)a, Kok & Vlassis (2006)] using coordination graphs. Figure 3.4 shows a graphical representation of a simple coordination graph for a 4-agent problem in a situation where an effect of an action taken by agent 3 depends on an action taken by agent 2 and an effect of an action taken by agent 2 depends on an action taken by agent 1. If an agent observes a state, variables and edges can be eliminated using some algorithms, then a complex coordination graph can be simplified resulting in reducing computational cost of coordination.

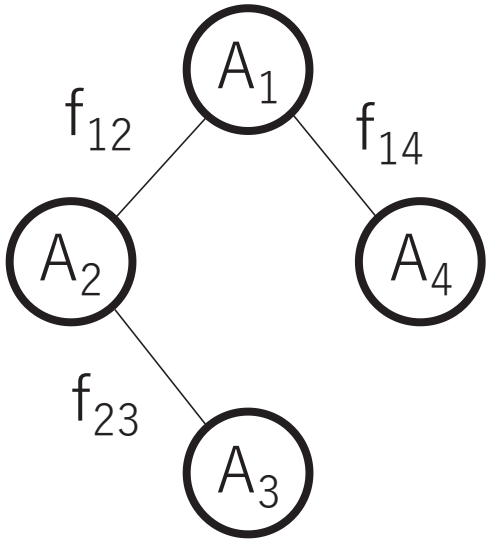


Fig. 3.4 A coordination graph for a 4-agent problem.

3.5.2 Utile Coordination

Kok et al. [Kok et al. (2005)] proposed Utile Coordination that is advanced method in which coordination graphs can be learned online instead of being designed manually

beforehand. Agents learned using Utile Coordination can learn context-specific dependencies that exist between specific agents using maintaining statistical information about obtained rewards conditioned on states and actions of other agents.

This method still has the problem of exponential increase in state-action space corresponding to the number of agents because it employs a complete multi-agent view of the entire joint state-action space to select joint actions even if the states of the other agents are not required to be considered.

3.5.3 Learning of Coordination

Melo and Veloso [Melo (2009)] suggested adding a pseudo-action, COORDINATE, to the action space of each agent. When COORDINATE is selected as an action by an agent, the agent obtains information about the other agents and behaves based on that information while the agent takes a penalty as a communication cost.

Because the Q-value for selecting COORDINATE can be obtained with Q-learning, the agent can decide when it should take the other agents into account. Setting the cost of COORDINATE for a specific game is a difficult issue because, if the cost is zero, agents will always choose COORDINATE, and, if the cost is too high, the agents will seldom choose it.

3.5.4 Entropy Based Approach

Arai and Xu [Arai & Xu (2016)] proposed an entropy-based method to distinguish interference states from others. Their approach consists of two processes.

In the first process the method detects interference states based on fluctuation of entropy of action policy π. The entropy $\mathrm{H}(\pi(s_h, A))$ is defined as following.

$$\mathrm{H}(\pi(s_h, A)) = -\sum_{i=1}^{n} \pi(s_h, a_i) \log \pi(s_h, a_i)$$

If an agent learns a deterministic policy, $\mathrm{H}(\pi(s_h, A))$ converges to zero for all the

state. In a multi-agent environment, agents can not learn a deterministic policy because the existence of other states affects what action should be selected in some states. In such states, the entropy does not converge but fluctuates during learning process. This method detects interference states by the fluctuation of the entropy.

In the second process agent i augments Q-values of the interference states from $Q_i(s_i, a_i)$ to $Q_i(s_i, \boldsymbol{s}_{-i}, a_i)$, where $\boldsymbol{s}_{-i}$ represents a combination of states of other agents than agent i. It selects Q-values based on whether it is in an interference state or not.

It is difficult for the method to deal with problems which involves more than two agents because the combination of the state of other agents $\boldsymbol{s}_{-i}$ includes both states, states related to the interference and states do not. The states which do not related to the interference may slow down the learning process.

3.5.5 CQ-learning

Hauwere et al. proposed CQ-learning [Hauwere (2010, 2011)]. A basic idea of CQ-learning is specifying states in which an agent must take a state of another agent into account by comparing averages of instant rewards between a single-agent environment and multi-agent environment. So that, CQ-learning requires prelearning in a single-agent environment to obtain averages of instant rewards for all the state-action pairs as well as Q-values for all the state-action pairs.

Once the difference is detected using Student's t-test, the state of agent k is augmented involving the state of another agent l. The agent decides its action based on Q-values of the augmented state $Q_k^{aug}(s_k, s_l)$ only if the agent and another agent are in the combination of state (s_k, s_l) while the agent usually decides its action based on Q-values of an unaugmented state $Q_k(s_k)$. Note that a state of another agent is used for the augmentation while the method mentioned in 3.5.3 uses combination states of all other agents for the augmentation.

The action is always selected ϵ-greedily. If action a_k is selected based on Q-values of

the augmented state $Q_k^{aug}(s_k, s_l)$, the Q-value $Q_k^{aug}(s_k, a_k, s_l)$ is updated using the same updating equation of Q-learning. Q-values of an unaugmented state $Q_k(s_k)$ are not updated.

The detailed algorithm of CQ-learning is described in Algorithm 3.1. $\mathrm{E}(R_k(s_k, a_k))$ stands for the expected values of the immediate rewards when agent *k* takes action a_k in state s_k in a single-agent environment, and $W_k(s_k, a_k)$ stands for samples of the immediate rewards when agent k takes action a_k in state s_k in a multi-agent environment.

Algorithm 3.1: CQ-learning algorithm for agent *k*

Train Q_k independently first, initialize Q_k^{aug} to zero and W_k=empty
Set t=0
for each episode **do**
  Initialize s_k
  for each step of episode **do**
    observe local state $s_k(t)$
    if $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ is present in the system state s(t) **then**
      Select $a_k(t)$ according to Q_k^{aug} ε-greedily
    else
      Select $a_k(t)$ according to Q_k ε-greedily
    end if
    observe $r_k = R_k(s(t), a(t))$, s'_k from T(s(t), a(t))
    Store $<s_k(t), a_k(t), r_k(t)>$ in $W_k(s_k, a_k)$
    if p-value of Student t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
      Store $<s_k(t), a_k(t), s_l(t), r_k(t)>$ in $W_k(s_k, a_k, s_l)$ for all other agents
      for all extra information s_i about another agent l present in s(t) **do**
        if p-value of Student t-test $(W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
          augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}
        end if
      end for
    end if
    if $s_k(t)$ is part of $\overrightarrow{s_k}$ and the information of $\overrightarrow{s_k}$ is in s(t) **then**

```
23:       Q_k^aug(s_k→, a_k) ← (1 − α_t)Q_k^aug(s_k→, a_k) + α_t[r_k + γmax_{a'_k} Q_k(s'_k, a'_k)]
24:     else
25:       No need to update Q-value
26:     end if
27:     t = t + 1
28:   end for
29: end for
```

Figure 3.5 shows an example of maze games in a multi-agent environment. Each agent tries to find an optimal path to finish the game, i.e. moving from their start locations to their goals. It is necessary to convert the maze game in the multi-agent environment to two maze games in a single-agent environment to apply CQ-learning. CQ-learning does not offer a specific method of the conversion. A problem in a multi-agent environment must be manually converted to corresponding problems in a single-agent environment.

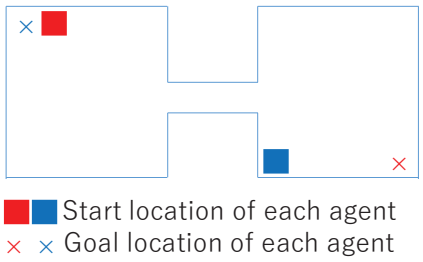


Fig. 3.5 An example of maze games.

Figure 3.6 shows two converted maze games in a single-agent environment. In maze games the converted maze games can be determined just by selecting each agent. Each agent gets the same penalties and rewards as the original maze game, i.e. -1 for a movement, -10 for a collision, and +0 for a goal.

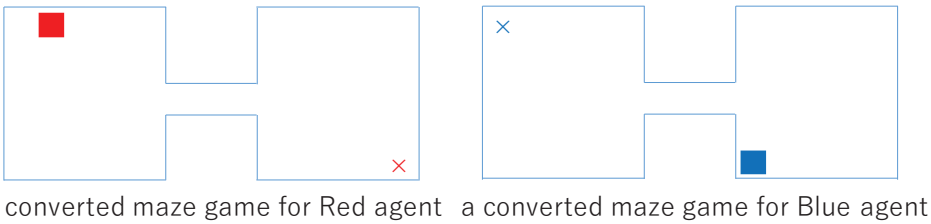


Fig. 3.6 Two examples of converted maze games.

Figure 3.7 illustrates the algorithm of CQ-learning. As shown in Fig 3.7 (a), Red agent using CQ-learning collects samples of rewards for every states and actions in the single-

agent environment. In the case of Fig. 3.7 (a), each reward of all the states and actions is -1 except for an action for the goal, i.e. +0.

In the multi-agent environment as shown in Fig. 3.7 (b), if Red agent collides with Blue agent, it gets -10 as a reward. When enough samples of rewards in the multi-agent environment are collected, i.e. 20+ samples, Student's t-test for the difference of average reward between the single-agent environment and the multi-agent environment is conducted. If the difference is detected by Student's t-test, the state of Red agent $(x_1, y_1) = (4, 2)$ is augmented to $(x_1, y_1, x_2, y_2) = (4, 2, 5, 2)$ involving the state of Blue agent $(x_2, y_2) = (5, 2)$ as shown in Fig. 3.7 (c).

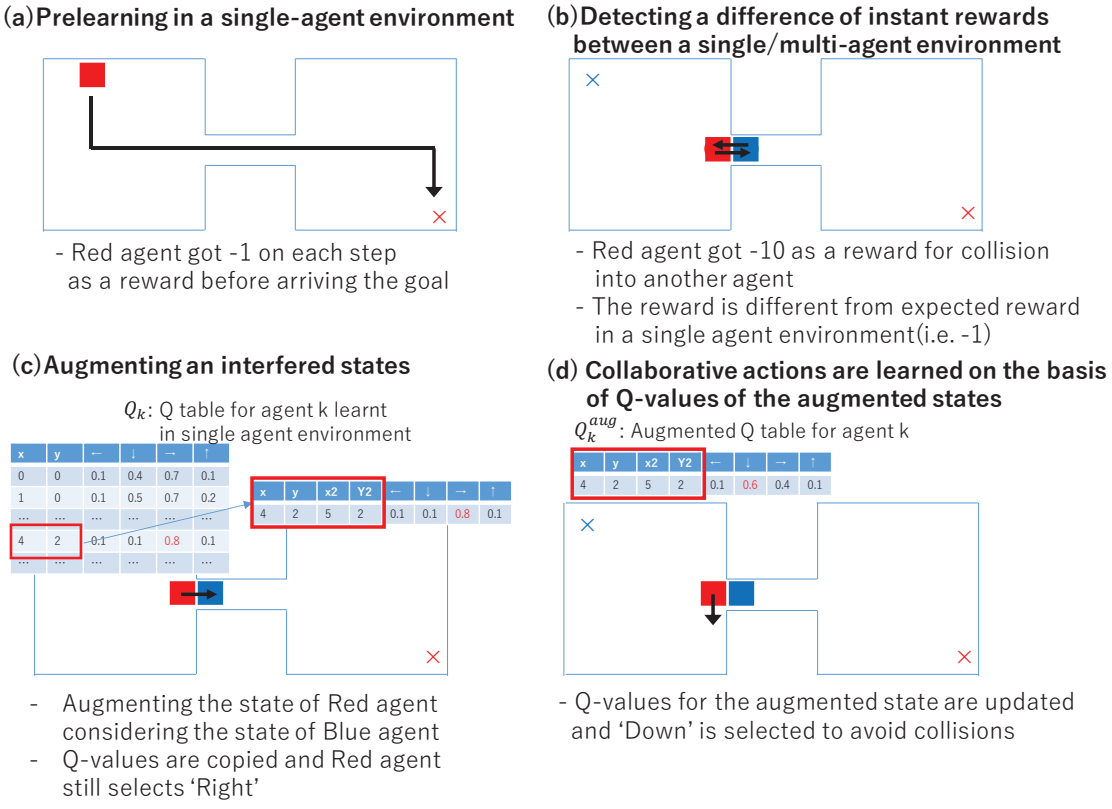


Fig. 3.7 Illustrated CQ-learning algorithm.

3.6 Summary

This chapter explains that there are some differences between a single-agent system and a multi-agent system. Dec-MDP can be used for describing a typical cooperative task in multi-agent systems. This thesis focuses on a special case of Dec-MDP, i.e. Dec-

SIMDP in which interaction among agents in an environment rarely happens. Some methods have been proposed to solve problems of Dec-SIMDP. They are different in how they detect when each agent should consider other agents but all of them augment the state of the agent once detecting it. The detailed algorithm of CQ-learning is described because the proposal method in this thesis is based on CQ-learning. In CQ-learning, a problem in a multi-agent environment are manually converted to multiple problems including each agent. After each agent learns the average reward for a specific action in a state in the converted problem in the single-agent environment, the agent statistically detects the difference of a reward for a specific action in a state between in the single-agent environment and in the multi-agent environment. Then the agent augments the state considering the state of another agent when the difference is detected.

Chapter 4 Enhancement of CQ-Learning

This thesis focuses on enhancing the performance of CQ-learning in a Dec-SIMDP. This chapter points out that CQ-learning has at least five issues and propose three methods, namely GCQ-learning, PCQ-learning, and GPCQ-learning, that improve a performance of CQ-learning by solving the issues. This chapter presents the result of evaluation of these methods using five maze games.

4.1 Issues of CQ-Learning

This chapter points out that CQ-learning has at least five issues to be addressed and improved as shown in Table 4.1 and describes each issue in detail.

Table 4.1 Issues of CQ-learning.

Issues	Description
Prelearning	How prelearning should be conducted?
Unnecessary Exploration	Selecting an action ϵ-greedily leads to unnecessary exploration.
Optimistic Q-values updating	Updating Q-values of an augmented states based on Q-values of an unaugmeted states is too optimistic.
Interference with more than two agents	How to choose one augmented state if an agent interferes with more than two agents?
Manual conversion of a problem	A problem in a multi-agent environment must be manually converted multiple problems in a single-agent environment.
Only detecting a difference of immediate rewards	CQ-learning only focus on a difference of immediate rewards between in a single-agent environment and in a multi-agent environment to detect interferences with another agent.

4.1.1 Prelearning

CQ-learning assumes that Q-values has already been converged to their optimal values by prelearning in a single-agent environment. However, they do not discuss how prelearing should be conducted.

In a maze game, an agent does not have to learn all the optimal Q-values for every state-action pair to find a shortest path to the goal. If the agent learns the optimal Q-values along an optimal path to the goal, it can achieve its objective: the smallest number of steps to the goal. On the other hand, the purpose of the prelearning of CQ-learning is to efficiently obtain the optimal Q-values for all the state-action pairs.

4.1.2 Unnecessary exploration

Because agents trained using CQ-learning always ϵ-greedily select an action, they may take random actions even if no interference with another agent is likely to occur. This causes unnecessary exploration and interferences in a multi-agent environment. In addition to that, taking a random action coincidently avoids interference with another agent, resulting in a lost opportunity for the agent to identify the difference between a single-agent environment and multi-agent environment.

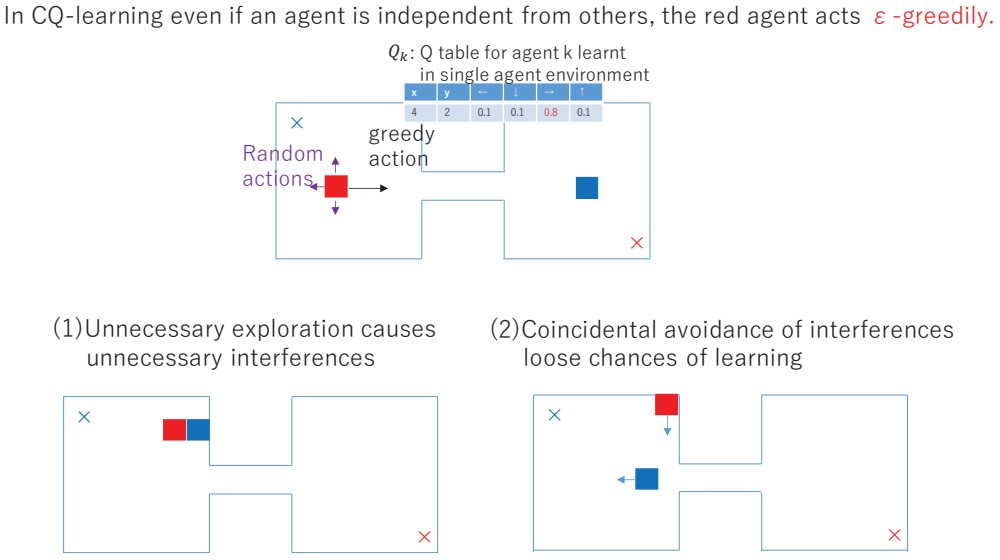


Fig. 4.1 Unnecessary exploration.

4.1.3 Optimistic Q-values updating

In CQ-learning Q-values learned in a single-agent environment, i.e., $Q_k(s'_k, a'_k)$, is used for the second item of the Q-values updating equation for the augmented state as follows.

$$Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1-\alpha_t)Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t[r_k + \gamma max_{a'_k} Q_k(s'_k, a'_k)]$$

This means that CQ-learning optimistically updates the Q-values of an augmented joint state assuming that after taking the selected action, the agent can behave based on independent Q-values without subsequent interference. This assumption is too optimistic because the distribution of interference states is localized and the probability of being still in another interference states for the agent after avoiding an interference is not neglectable. In Fig. 4.2 a red agent avoid collision by selecting its action based on its augmented Q-values. CQ-learning assumes that the agent can independently select its action because it has already avoided the collision. However, it is likely that the agent collides with the same agent because the agent is still in the near location.

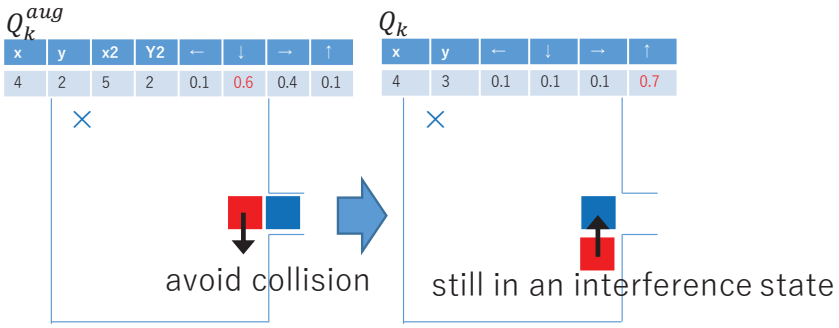


Fig. 4.2 An example of subsequent interferences.

4.1.4 Interference with more than two agents

CQ-learning uses Q-values of an augmented states if another agent is in an interference state. The last issue is that CQ-learning does not specify which augmented state should be used if more than two agents are in interference states at the same time.

As shows in Fig. 4.3, a red agent is next to two other agents. If both combination of the state of the red agent and the state of another agent has been augmented, one augmented

Q-values must be used to decide the action of the red agent. If the combination with the right agent is selected, the red agent selects an action *down* while it selects an action *right* if the combination with the upper agent is selected as shown in Fig. 4.3. Because CQ-learning does not specify which Q-values of augmented states should be used in this case, we need to have a way to select a specific Q-values from multiple candidates.

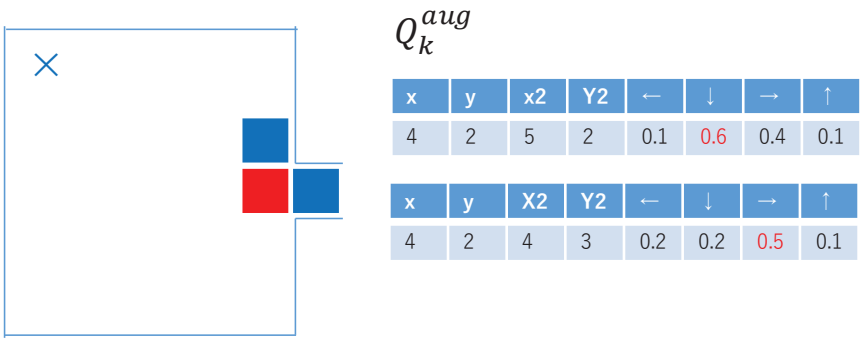


Fig. 4.3 An example of interferences with more than two agents.

4.1.5 Manual conversion of a problem

As mentioned in Section 3, CQ-learning does not provide any specific methods to convert a problem in a multi-agent environment to multiple problems in a single-agent environment. Although the conversion is intuitive and simple in maze games, it is not assured that the conversion is manually and easily done for practical problems. In this thesis, the issue is not solved and be remained as a future work.

4.1.6 Only detecting a difference of immediate rewards

If an agent gets -1 for a collision, CQ-learning does not detect any differences of immediate rewards even if collisions occurs. Although there is a difference of cumulative rewards to finish the game due to interferences between agents, CQ-learning does not consider it. As mentioned in Section 3, there are some methods that detect interferences based on a different perspective. This thesis does not cover different perspectives than a difference of immediate rewards to detect interferences between agents.

4.2 Solving Issues

This section proposes four approaches to solve the issues mentioned in subsection 4.1.1-4.1.4. The issues mentioned in 4.1.5 and 4.1.6 are out of scope in this thesis.

4.2.1 Prelearning

The factor of random action selection ϵ affects the efficiency of reinforcement learning because it controls a balance between exploration of an environment and exploitation of learnt knowledge about the environment. If we set ϵ close to 1.0, an agent randomly selects its action resulting in longer steps to reach its goal while the agent can learn more about the environment. If we set ϵ close to 0.0, the agent tends to greedily select its action and loses chances to learn about the environment. If the agent loses chances to learn about the environment, estimated $Q^*(s,a)$ likely has much error to real $Q^*(s,a)$.

Figure 4.4 shows RMSE between estimated $Q^*(s,a)$ and the real $Q^*(s,a)$ corresponding to ϵ values. If ϵ is near to zero, it takes longer episodes to learn Q-values close to $Q^*(s,a)$. If we set ϵ to higher value like 0.8, estimated $Q^*(s,a)$ is rapidly converged to the real $Q^*(s,a)$. However, if we set ϵ to almost 1.0, it takes much longer episodes to converge estimated $Q^*(s,a)$. This thesis uses ϵ= 0.8 in this thesis to maintain low RMSE between the estimated $Q^*(s,a)$ and the real $Q^*(s,a)$ and high learning efficiency.

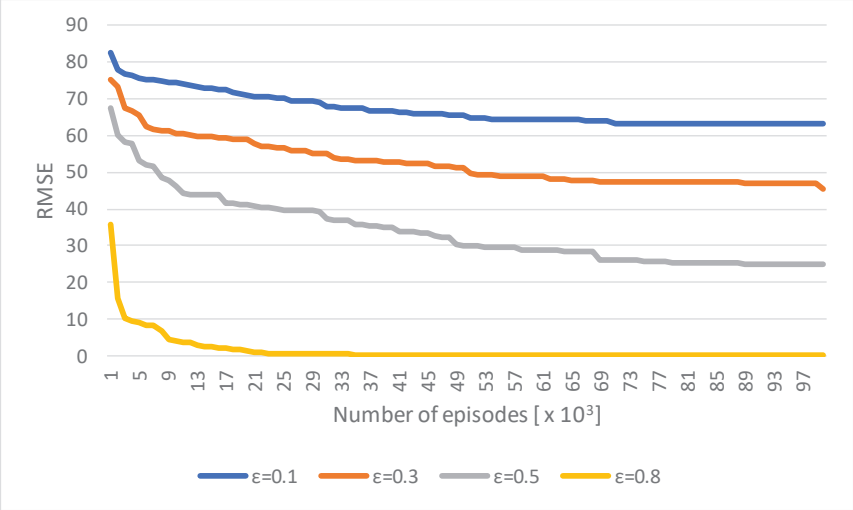


Fig. 4.4 RMSE between Q and Q^*

4.2.2 Greedy action selection

This thesis proposes that an agent should greedily select its action if the agent is not in an interference states because the agent has already learnt how it should select its action independently and ϵ-greedily action selection may leads to unnecessary exploration.

4.2.3 Pessimistic Q-value updating

As shown in Fig. 4.2, because the probability of subsequent interference with another agent is not neglectable. In addition to that, when Q-values are to be updated, whether it interferes with another agent again or not is fixed.

That is why this thesis proposes selecting Q-value updating equation based on whether the agent is still in an interference state after the action.

If agent k selects action a_k and is still in an interference states (s'_k, s'_l) with agent l, Q-value is updated on the basis of $Q_k^{aug}(s'_k, a'_k, s'_l)$, which means the next actions selection will be selected considering the state of another agent.

$$Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1-\alpha_t)Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t[r_k + \gamma max_{a'_k} Q_k^{aug}(s'_k, a'_k, s'_l)]$$

If s'_k is not an interference state, Q-value is updated on the basis of $Q_k(s'_k, a'_k)$, which

means the actions selection will be selected independently.

$$Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1-\alpha_t)Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t[r_k + \gamma max_{a'_k} Q_k(s'_k, a'_k)]$$

4.2.4 Interference with more than two agents

One approach to deal with a situation in which more than two agents interfere with each other is to augment the state of the agents involving all the states of others. For example, the state of agent k can be augmented to $Q_k^{aug}(s_k, s_l, s_m)$ if agent k, l, and m are in an interfere states s_k. Although this approach solves the issue of selecting Q-values from $Q_k^{aug}(s_k, s_l)$ and $Q_k^{aug}(s_k, s_m)$, it raises more serious problem; explosion of augmented states.

Another approach is to select a specific augmented state from all the states involving in an interference among more than two agents based on a specific criterion. However, it is difficult to design an adequate criterion for the selection. This thesis simply proposes randomly selecting an augmented state.

4.3 Proposed Methods

This thesis has now proposed three methods, namely GCQ-, PCQ-, and GPCQ-learning, to improve the performance of CQ-learning by solving issues as mentioned in section 4.2 [Kujirai & Yokota (2018, 2019)].

Table 4.2 shows relationship between our proposed methods and issues.

Prelearnig approach and interference with more than two agents approach are exploited for all the proposed methods. GCQ-learning exploits greedy action selection approach, PCQ-learning exploits pessimistic Q-value updating approach, and GPCQ-learning exploits both.

Table 4.2 Relationship between proposed methods and issues.

	Prelearning	Greedy action Selection	Pessimistic Q-value updating	Interference with more than two agents
GCQ-learning	x	x		x
PCQ-learning	x		x	x
GPCQ-learning	x	x	x	x

4.3.1 GCQ-learning

GCQ-learning exploits the greedy action selection approach. Detailed algorithm of GCQ-learning is described in Algorithm 4.1. Differences from CQ-learning are specified by hatching.

Algorithm 4.1: GCQ-learning algorithm for agent k

1:Train Q_k independently first, initialize Q_k^{aug} to zero and W_k=empty
2:Set t=0
3:**for** each episode **do**
4: Initialize s_k
5: **for** each step of episode **do**
6: observe local state $s_k(t)$
7: **if** $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ present in the system state s(t) **then**
8: **if** a set of $\overrightarrow{s_k}$ contains more than two $\overrightarrow{s_k}$ with $s_k(t) = s_k$ **then**
9: Select an agent l randomly from a set of $\overrightarrow{s_k}$ with $s_k(t) = s_k$
10: Select $a_k(t)$ in accordance with $Q_k^{aug}(s_k, s_l)$ ε-greedily
11: **else**
12: Select $a_k(t)$ in accordance with Q_k^{aug} ε-greedily
13: **end if**
14: **else**
15: Select $a_k(t)$ in accordance with Q_k greedily
16: **end if**
17: observe $r_k = R_k(s(t), a(t))$, s'_k from T(s(t), a(t))
18: Store $<s_k(t), a_k(t), r_k(t)>$ in $W_k(s_k, a_k)$

19: **if** p-value of Student's t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k))$ $< p_{th}$ **then**

20: Store $<s_k(t), a_k(t), s_l(t), r_k(t)>$ in $W_k(s_k, a_k, s_l)$ for all other agents

21: **for** all extra information s_i about another agent l Present in s(t) **do**

22: **if** p-value of Student's t-test $W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**

23: augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}

24: **end if**

25: **end for**

26: **end if**

27: **if** $s_k(t)$ is part of $\overrightarrow{s_k}$ and information of $\overrightarrow{s_k}$ is in s(t) **then**

28: $Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1-\alpha_t) Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t [r_k + \gamma max_{a_k'} Q_k(s_k', a_k')]$

29: **else**

30: No need to update Q-value

31: **end if**

32: t=t+1

33: **end for**

34: **end for**

4.3.2 PCQ-learning

PCQ-learning exploits the pessimistic Q-value updating approach. Detailed algorithm of PCQ-learning is described in Algorithm 4.2. Differences from CQ-learning are specified by hatching.

Algorithm 4.2: PCQ-learning algorithm for agent k

1:Train Q_k independently first, initialize Q_k^{aug} to zero and W_k=empty

2:Set t=0

3:**for** each episode **do**

4: Initialize s_k

5: **for** each step of episode **do**

6: observe local state $s_k(t)$

7: **if** $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ present in the system state s(t) **then**

8: **if** a set of $\overrightarrow{s_k}$ contains more than two $\overrightarrow{s_k}$ with $s_k(t) = s_k$ **then**

Select an agent l randomly from a set of $\overrightarrow{s_k}$ with $s_k(t) = s_k$
Select $a_k(t)$ in accordance with $Q_k^{aug}(s_k, s_l)$ ε-greedily
else
Select $a_k(t)$ in accordance with Q_k^{aug} ε-greedily
end if
else
Select $a_k(t)$ in accordance with Q_k ε-greedily
end if
observe $r_k = R_k(s(t), a(t))$, s'_k from $\mathrm{T}(\mathrm{s}(\mathrm{t}), \mathrm{a}(\mathrm{t}))$
Store $<s_k(t), a_k(t), r_k(t)>$ in $W_k(s_k, a_k)$
if p-value of Student's t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k))$ $< p_{th}$ **then**
Store $<s_k(t), a_k(t), s_l(t), r_k(t)>$ in $W_k(s_k, a_k, s_l)$ for all other agents
for all extra information s_i about another agent l Present in $\mathrm{s}(\mathrm{t})$ **do**
if p-value of Student's t-test $W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}
end if
end for
end if
if $s_k(t)$ is part of $\overrightarrow{s_k}$ and information of $\overrightarrow{s_k}$ is in $\mathrm{s}(\mathrm{t})$ **then**
if s'_k and s'_l is in part of $\overrightarrow{s_k}$ **then**
$Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1 - \alpha_t) Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t [r_k + \gamma max_{\alpha'_k} Q_k^{aug}(s'_k, a'_k, s'_l)]$
else
$Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1 - \alpha_t) Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t [r_k + \gamma max_{\alpha'_k} Q_k(s'_k, a'_k)]$
end if
else
No need to update Q-value
end if
t=t+1
end for
end for

4.3.3 GPCQ-learning

GPCQ-learning exploits both the greedy action selection approach and the pessimistic Q-value updating approach. Detailed algorithm of GPCQ-learning is described in Algorithm 4.3. Differences from CQ-learning are specified by hatching.

Algorithm 4.3: GPCQ-learning algorithm for agent k

1:Train Q_k independently first, initialize Q_k^{aug} to zero and W_k=empty
2:Set t=0
3:**for** each episode **do**
4: Initialize s_k
5: **for** each step of episode **do**
6: observe local state $s_k(t)$
7: **if** $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ present in the system state s(t) **then**
8: **if** a set of $\overrightarrow{s_k}$ contains more than two $\overrightarrow{s_k}$ with $s_k(t) = s_k$ **then**
9: Select an agent l randomly from a set of $\overrightarrow{s_k}$ with $s_k(t) = s_k$
10: Select $a_k(t)$ in accordance with $Q_k^{aug}(s_k, s_l)$ ε-greedily
11: **else**
12: Select $a_k(t)$ in accordance with Q_k^{aug} ε-greedily
13: **end if**
14: **else**
15: Select $a_k(t)$ in accordance with Q_k greedily
16: **end if**
17: observe $r_k = R_k(s(t), a(t))$, s'_k from T(s(t), a(t))
18: Store $<s_k(t), a_k(t), r_k(t)>$ in $W_k(s_k, a_k)$
19: **if** p-value of Student's t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
20: Store $<s_k(t), a_k(t), s_l(t), r_k(t)>$ in $W_k(s_k, a_k, s_l)$ for all other agents
21: **for** all extra information s_i about another agent l Present in s(t) **do**
22: **if** p-value of Student's t-test $W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
23: augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}
24: **end if**
25: **end for**
26: **end if**
27: **if** $s_k(t)$ is part of $\overrightarrow{s_k}$ and information of $\overrightarrow{s_k}$ is in s(t) **then**

```
    if s'_k and s'_l is in part of s_k→ then
      Q_k^aug(s_k→, a_k) ← (1 − α_t)Q_k^aug(s_k→, a_k) + α_t[r_k + γmax_{α'_k} Q_k^aug(s'_k, a'_k, s'_l)]
    else
      Q_k^aug(s_k→, a_k) ← (1 − α_t)Q_k^aug(s_k→, a_k) + α_t[r_k + γmax_{α'_k} Q_k(s'_k, a'_k)]
    end if
  else
    No need to update Q-value
  end if
  t = t + 1
 end for
end for
```

4.4 Evaluation

This section evaluates the proposed GCQ-, PCQ-, and GPCQ-learning methods using five maze games shown in Fig. 3.1.

As shown in Table 4.3, the number of state-action combinations increases exponentially with the number of states and agents with JSQ-learning (JSQ) and JSAQ-learning (JSAQ). In contrast, the initial number of state-action combinations of CQ-learning and the proposed methods (referred to as SI(sparse interaction) in the table) is equal to that of independent learning and increases linearly with the number of states and agents. The number of augmented joint states will be discussed later in this section.

Table 4.3 Number of state-action combination in maze games.

	No. of states	Independent	JSQ	JSAQ	SI
TunnelToGoal	25	200	5,000	10,100	200
ISR	43	244	14,792	29,584	244
CIT	69	552	38,088	76,176	552
CMU	133	1,064	141,512	283,024	1,064
TunnelToGoal3	55	660	1,996,500	10,648,000	660

For the agents trained using CQ-learning and our proposed learning methods, the length of the window used to calculate the distribution of immediate rewards was set to 20. Student's t-test was performed for the distribution between the immediate rewards in a single-agent environment and in a multi-agent environment only when 20+ immediate reward samples were obtained for a state in a multi-agent environment. The threshold of the t-test, p_{th}, was set to 0.01, as was done by Hauwere et al. [Hauwere (2010, 2011)]. Only when the null hypothesis (the mean of the immediate reward samples in a certain state in a multi-agent environment is the same as the expected immediate rewards in the same state in a single-agent environment) was rejected was the state augmented because it could be an interference state.

CQ-learning and the proposed learning methods require prelearning of independent Q-values for each agent. For each game, all the agents first learned the Q-values ϵ -greedily in a single-agent environment for 10,000 episodes with ϵ =0.8 to ensure that the agents could sufficiently explore the environment. An agent using CQ-learning and the proposed methods uses independent Q-values to select an action. Because all the mazes were designed so that the agents would interfere with each other if they selected their actions based on independent Q-values learned in a single-agent environment, the agents were likely to collide as they made their way towards their goals.

The number of episodes was set to 20,000 for independent learning, JSQ-learning, and JSAQ-learning and 10,000 for CQ-, GCQ-, PCQ-, and GPCQ-learning because CQ-learning and its extensions require prelearning (in this case, 10,000 episodes) in a single-agent environment.

Each method is evaluated from perspectives of number of average steps to goal, number of average augmented states, and computational time.

4.4.1 Number of average steps to goal

The results shown in Table 4.4 include the number of average steps to the goal, the standard deviation during the last 100 episodes in 10,000 episodes. The light gray cells

indicate the methods that resulted in the smallest number of steps to the goal.

The agents trained using independent-learning had difficulty learning the optimal action policy for all the games.

The agents trained using joint-state learning had difficulty learning the optimal action policy for the CMU and TunnelToGoal3 games when the number of episodes was limited to 10,000 due to a large number of state-action combinations.

If an agent trained using JSAQ-learning, which practically controls all the agents, had sufficient time to explore all the state-action space many times thoroughly, it was able to learn the optimal joint action policy. It had trouble doing this in our evaluation experiment because of the exponentially increasing state-action space in the CMU and TunnelToGoal3 games (Table 4.3).

The agents trained using CQ-learning did not have the best performance for any game and had the second-best performance only for the TunnelToGoal3 game.

The agents trained using GCQ-learning did not have the good performance for any game. They exhibited unstable behavior in the ISR and CIT games, resulting in a deviation in the path taken to the goal. This is because they took greedy actions based on the independent Q-values after they avoided a collision, which resulted in subsequent collisions.

The agents trained using PCQ-learning had the best performance only for the TunnelToGoal3 game and had the second-best performance only for the ISR game.

The agents trained using modified GPCQ-learning, which uses both methods, achieved the best performance in terms of the average number of steps to the goal in the TunnelToGoal, ISR, CIT, and CMU games. This is attributed to their ability to find the differences between the single-agent environment and the multiagent environment as well as to estimate the probability of sequential interference.

In the TunnelToGoal3 game, the performances of all the methods were far from being optimal. Even PCQ-learning did not exhibit improved performance because the maze does not conform to the assumption of sparse interaction as the agents frequently interfered with each other near the entrance of the tunnel to the goal.

Table 4.4 Number of average steps to goal and the standard deviation.

	Tunnel2Goal		ISR		CIT		CMU		TunnelToGoal3	
	mean	std	mean	std	mean	std	mean	std	mean	std
Min Steps	11		6		13		33		12	
Indep20000	21.9	25.0	13.3	29.5	26.5	2.77	44.4	4.31	24.7	15.7
JSQ20000	14.8	5.00	8.89	10.5	21.4	19.9	167	69.8	212	188
JSAQ20000	12.7	7.77	11.9	62.9	32.1	73.7	740	351	532	418
CQ	13.5	3.84	31.6	79.9	25.7	26.9	41.7	8.58	23.8	19.2
GCQ	13.2	4.87	51.2	229	32.9	133	33.8	5.01	28.2	16.7
PCQ	12.8	1.59	8.64	5.87	19.1	15.2	39.4	4.56	23.1	17.7
GPCQ	11.3	0.616	7.42	1.68	15.6	7.68	33.3	2.94	30.9	22.6

Figure 4.5 visualizes the number of average steps to goal and the deviations given in Table 4.4. Note the negative sides of the standard deviations are omitted in the figure.

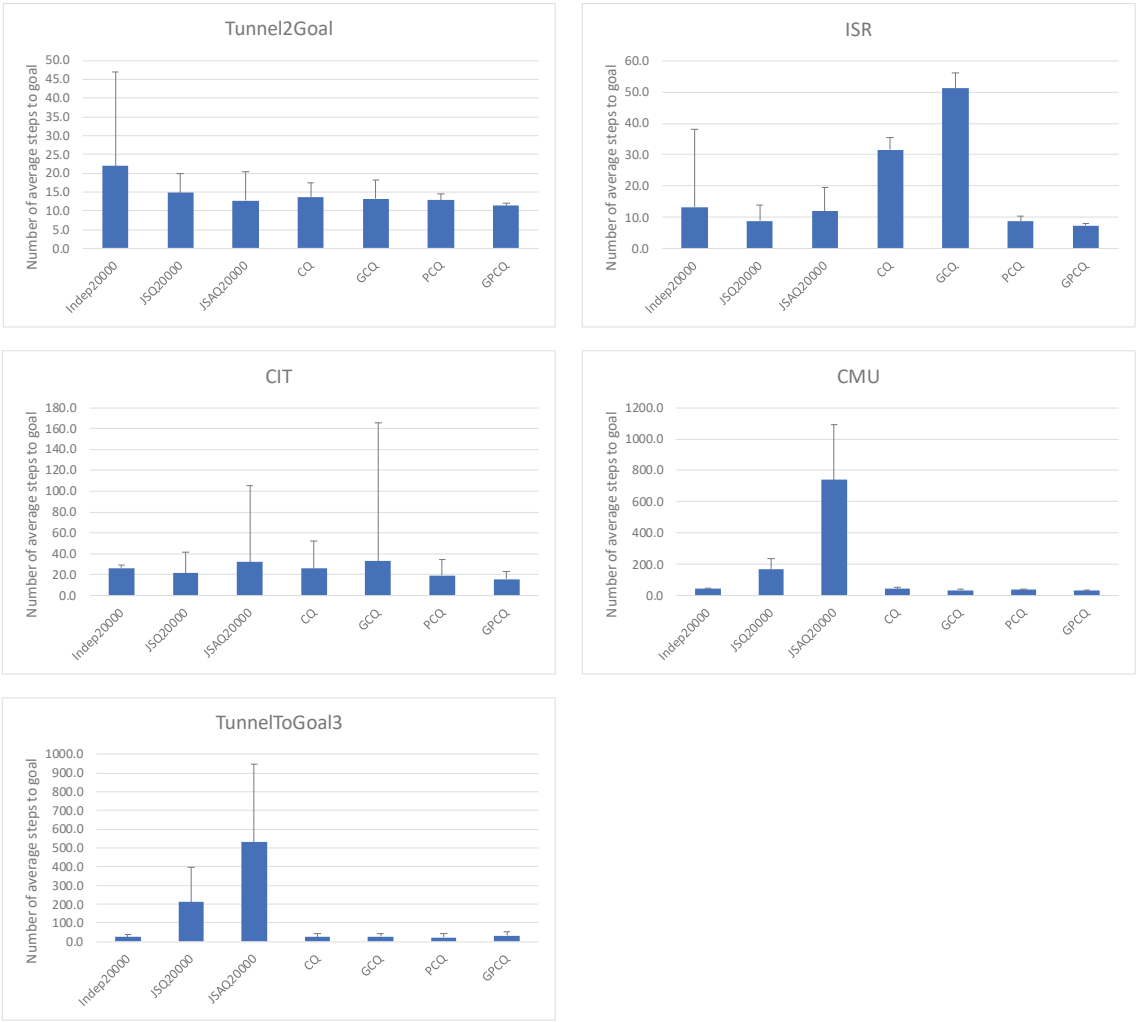


Fig. 4.5 Comparison of methods in number of average steps to goal.

Figure 4.6 shows the learning curves of each method, i.e., CQ-, GCQ-, PCQ-, and GPCQ-learning, in the maze games.

In the TunnelToGoal game, each method almost converged in number of steps to goal. GPCQ-learning had quickly converged to the smallest number of steps to goal.

In the ISR and CIT game, CQ- and GCQ-learning first had a difficulty to find a good path to goal and slowly learned better paths. PCQ-learning had better performance and quickly converged while GPCQ-learning had the best performance.

In the CMU game, PCQ-learning had the second worst performance because in the game, considering sequential collisions are not necessary and one of the agents just let another agent go by detouring just once.

In the TunnelToGoal3 game, all the method had a difficulty to find the shortest path, i.e., 12 steps. Although CQ- and PCQ-learning first had better performance, their performances were degraded in learning process due to exploration by ϵ-greedily action selection.

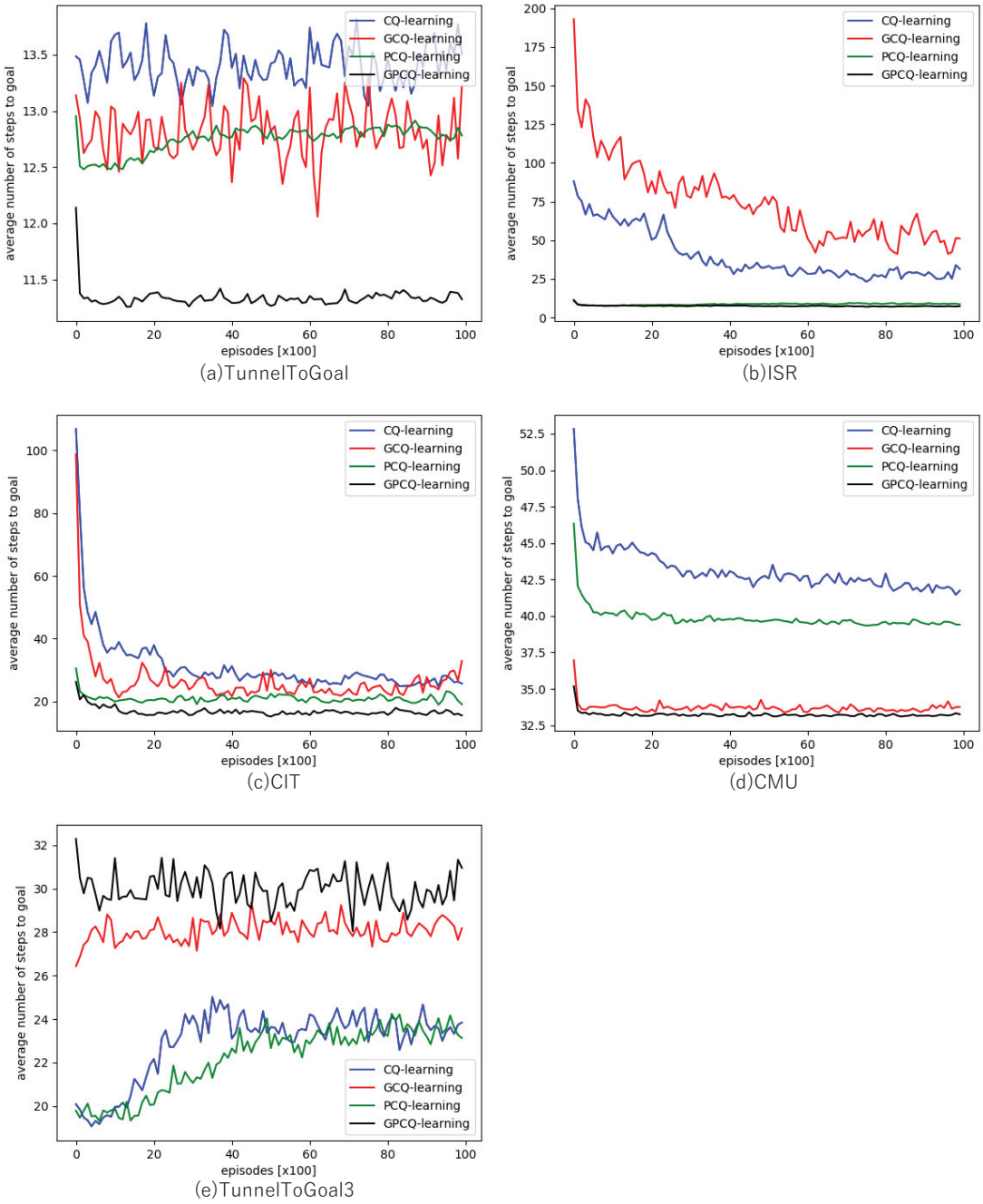


Fig. 4.6 Learning curves of each method in maze games.

4.4.2 Number of average augmented states

Table 4.5 shows the results of the average number of augmented states after 10,000 episodes. Number of augmented states in GCQ- and GPCQ-learning was smaller than in CQ- and PCQ-learning for most of the games while GCQ-learning had poorer

performances and GPCQ-learning had better performances. It indicates that the agents trained using GCQ-learning can not sufficiently explore the state-action combinations when a good route is not found due to the greedy action selection while the agents trained using GPCQ-learning can augment at least necessary states.

Table 4.5 Number of augmented states.

	Tunnel2Goal		ISR		CIT		CMU		TunnelToGoal3	
	mean	std	mean	std	mean	std	mean	std	mean	std
CQ	8.76	2.62	11.4	2.00	31.1	3.03	41.3	6.15	37.7	6.30
GCQ	2.96	0.20	12.7	1.22	25.0	3.22	12.8	3.72	13.7	0.82
PCQ	9.18	1.02	13.0	3.17	26.9	4.52	44.7	4.00	40.0	3.52
GPCQ	2.76	0.476	12.0	2.05	20.5	2.61	11.5	2.70	13.7	1.45

Figure 4.7 shows an example of augmented joint states in the TunnelToGoal game with agents trained using GPCQ- and CQ-learning. The circled numbers represent the locations of the agents where joint states were augmented in the state space of agent 1.

As shown in the figure, augmented joint states can be categorized as augmented by collision and augmented by waiting. In an augmented by collision state, if both agents move in the directions indicated by the arrows (i.e., (c-1), (c-2), and (c-3)) or an agent moves in a direction indicated by an arrow and the other does not move (i.e., (c-4) and (c-5)), a collision occurs, and each agent receives a reward of –10. In a single-agent environment, the agent received a reward of –1 for the same action taken in the same state. The difference in rewards is detected using Student's t-test, and the state is augmented. Augmented by waiting has a different mechanism. In a single-agent environment, the agent receives a reward of 0 simply for reaching the goal. In contrast, in a multi-agent environment, even if an agent selects the same action, it receives a reward of –1 if another agent has not yet reached the goal. In this case, whatever action agent 2 takes at any position (i.e., (w-1) and (w-2)), a difference in immediate rewards between the two environments is observed.

As shown in Fig. 4.7, the number of augmented joint states for both categories is higher with CQ-learning than with GPCQ-learning. This indicates that ϵ-greedy action selection in CQ-learning leads to unnecessary exploration.

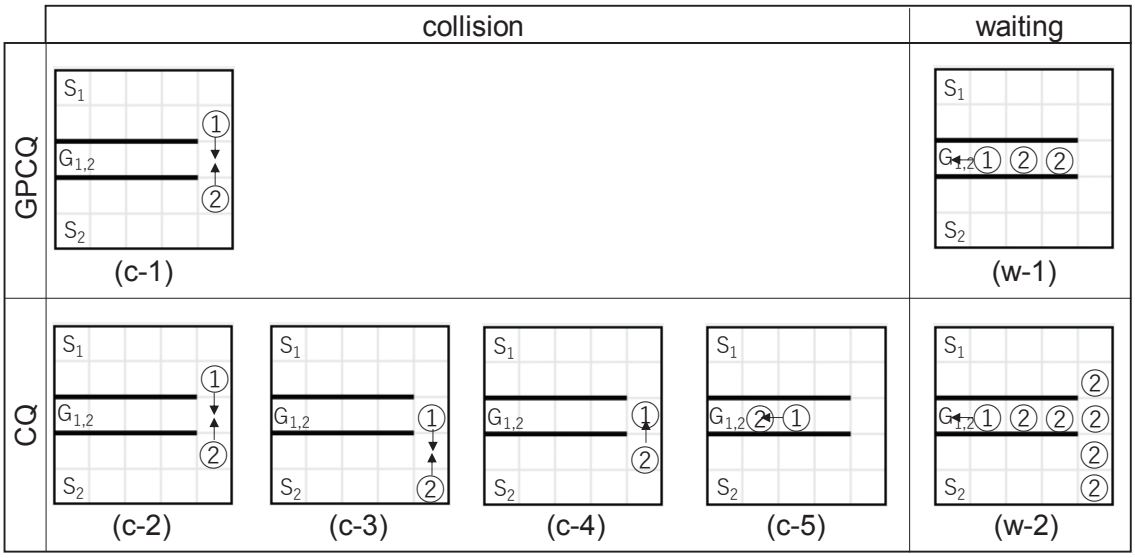


Fig. 4.7 An example of augmented joint states in the TunnelToGoal game.

Figure 4.8 shows examples of the paths taken when each agent selected greedy actions based on the action policies learned using JSQ-, JSAQ-, CQ-, and our proposed GPCQ-learning in the TunnelToGoal game. The actions of agent 1 are depicted with solid lines, and those of agent 2 are depicted with dotted lines. Thin arrows represent actions taken base on Q_k, and thick arrows present actions taken based on Q_k^{aug}. In all the games, agent 2, which started from S_2, took the shortest path to the goal in the examples. In contrast, agent 1, took a detour route, thereby avoiding a collision with agent 2. The agents trained using JSQ- and JSAQ-learning did not take the optimal path. Although the agents trained using the CQ- and GPCQ-learning learned the optimal action policy, those trained using CQ-learning augmented more joint states (i.e., 7, 8, and 9 in the black circles), so more episodes were required for the agents to learn the optimal Q-values for the augmented joint states.

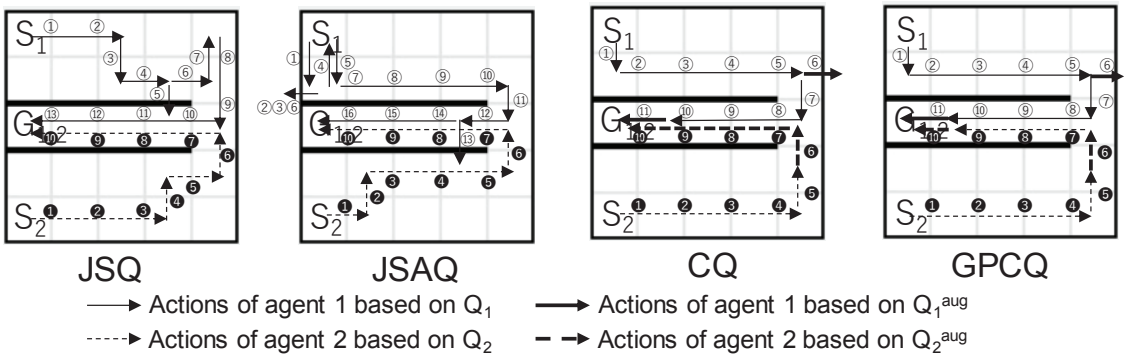


Fig. 4.8 Examples of paths taken on basis of learned action policy in the TunnelToGoal game.

4.4.3 Computational time

None of the proposed methods require extra computational cost other than the cost for selecting the Q-values and the equations for updating Q-values, as shown in Algorithm 4.2-4.4. For example, the average computational time for each step in the TunnelToGoal game was 0.346 ms for CQ-learning and 0.330 ms for GPCQ-learning as shown in Fig. 4.9. GPCQ-learning required less computational time for each step because it augmented fewer joint states than CQ-learning, as shown in Table 4.5. This reduced the computational cost of updating the Q-values for the augmented joint states.

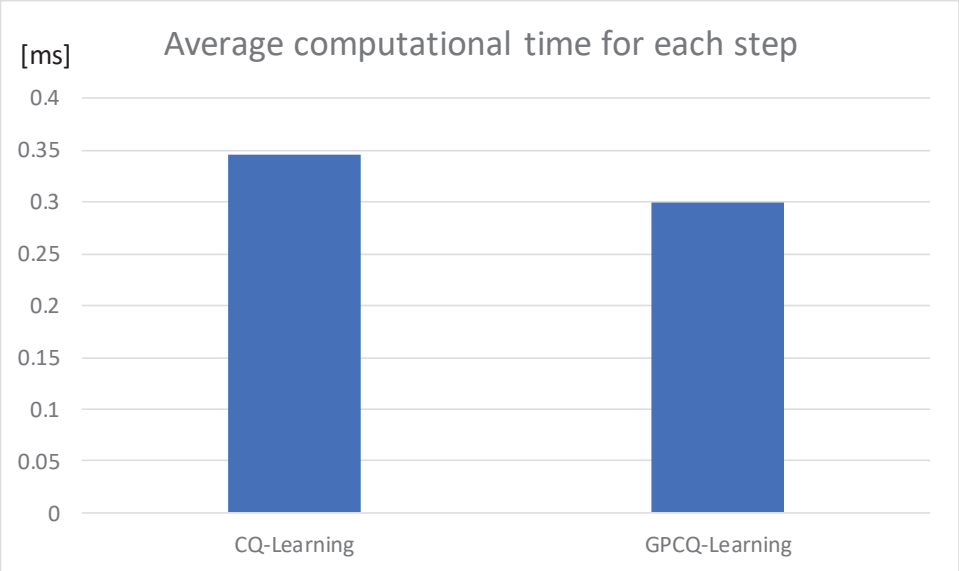


Fig. 4.9 Average computational time for each step.

4.5 Summary

This chapter points out the four issues CQ-learning has and described approaches to solve them. This chapter then proposes three methods that solve some of the issues; namely GCQ-, PCQ-, and GPCQ-learning. Evaluation using five maze games demonstrated that GPCQ-learning that exploits all the four approaches had the best performance in most of the maze game except for the TunnelToGoal3 in which assumption of sparse interaction does not hold.

Chapter 5 Enhancement of GPCQ-Learning

This chapter points out that the agents trained using GPCQ-learning sometimes fall into a deadlock because of greedy action selection and no difference of instant rewards between in a single-agent environment and in a multi-agent environment. An agent that takes an action based on an augmented states moves to an unaugmented state, then always selects an action to move back to the same state due to greedy action selection in the unaugmented state. If the reward of the action is the same as that of the action at the same state in the single-agent environment, the agent can not augment the state, resulting in a deadlock.

This chapter describes how such a deadlock occurs in detail and proposes two methods to solve the deadlock.

5.1 Pursuit Games

In pursuit games, agents (depicted by numbers in Fig. 3) move to “touch” a target (depicted by T) in a square field, and the game finishes when all the agents are touching the target. In this thesis, the target does not move from the initial position.

Rewards are designed as -1 for a movement, -10 for a collision with another agent, and 0 for finish.

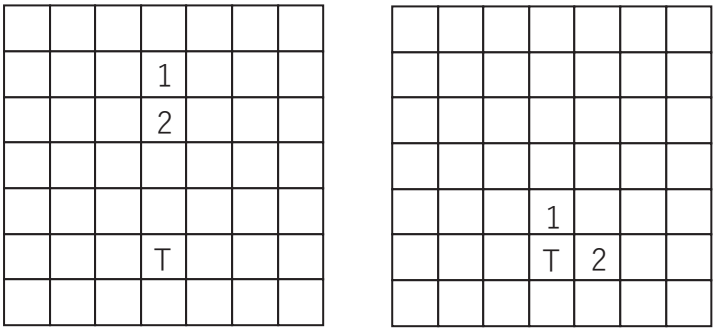


Fig. 5.1 An example of pursuit games.

First, an agent is trained to learn how to move in order to touch the target in a single-agent environment. The conversion to problems in a single-agent environment is manually designed. The initial positions of the target and the agent are randomly selected in every episode.

Then, multiple agents try to find an optimal policy for moving in order to touch the target at the same time in a multi-agent environment. The initial positions of the target and the agents were fixed in seven patterns for evaluation, as shown in Fig. 5.2. For patterns 1−3, the agents can touch the target by greedily selecting their actions without interference with the other agents. For patterns 4−7, an agent collides with another agent if it greedily selects its action based on Q-values learned in a single-agent environment.

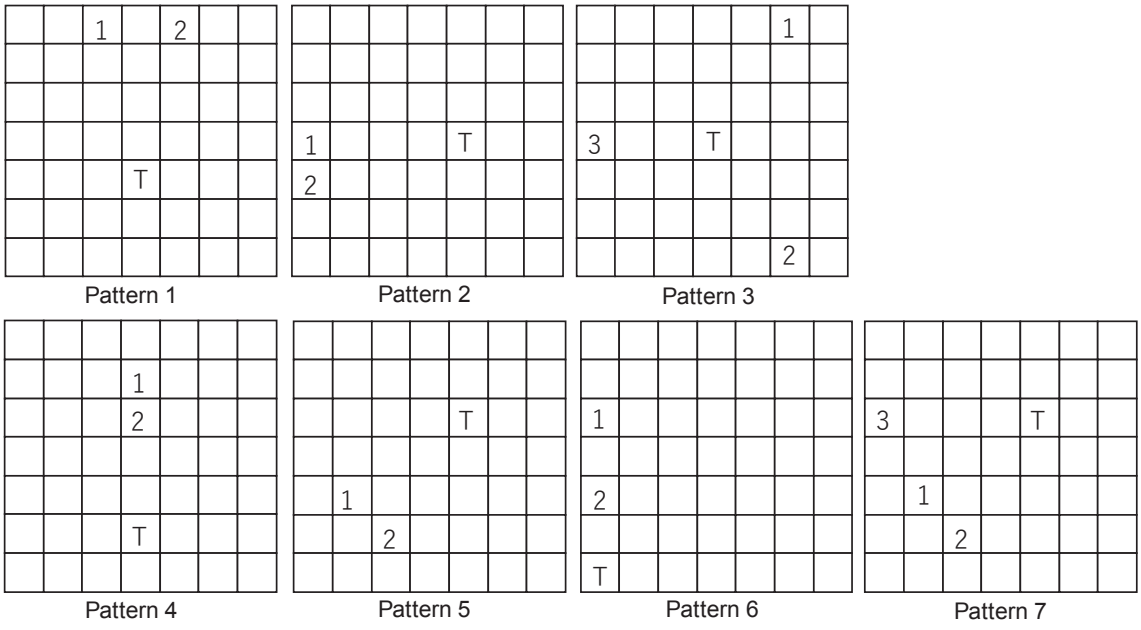


Fig. 5.2 Seven initial agent/target-position patterns used for evaluation.

5.2 Deadlocks in Pursuit Games

The performances of CQ- and GPCQ-learning using seven initial agent/target-position patterns shown in Fig. 5.2 are evaluated and shown in Table 5.1.

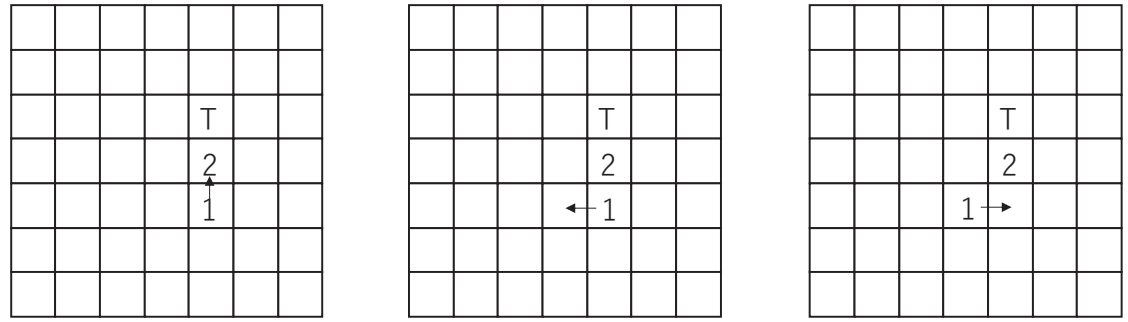
For patterns 1−3, the agents using GPCQ-learning find optimal paths resulting in the

minimum number of steps to finish while agents using CQ-learning take more steps because they ϵ-greedily select their actions, resulting in unnecessary exploration. For pattern 4, the agents using GPCQ-learning perform better because they avoid unnecessary augmentation of joint states. For patterns 5−7, the agents using CQ-learning perform much better. Patterns 5−7 in paticular, the agents using GPCQ-learning rarely finish the games (depicted as – in Table 2). This is because a repetitive pattern of action-states does not create a difference in the immediate rewards, as shown in Fig. 5.3.

Table 5.1 Comparison of CQ and GPCQ-learning for pursuit games.

				CQ		GPCQ	
Patterns	Interference	No. of agents	Min No. of steps	Mean	Std dev.	Mean	Std dev.
1	No	2	4	5.15	1.73	4.00	0.00
2	No	2	4	6.45	8.88	4.00	0.00
3	No	3	4	6.26	13.7	4.00	0.00
4	Yes	2	4	5.87	1.57	5.12	0.482
5	Yes	2	3	18.8	28.2	–	–
6	Yes	2	4	178	291	–	–
7	Yes	3	5	30.4	54.0	–	–

Looking at Fig. 5.3, we see that agent 1 first greedily selects an action of *upward* in accordance with the prelearned Q-value and detects a difference in immediate rewards because it collides with agent 2 (Fig. 5 (a)). It then detects a difference in immediate rewards and augments its state with the state of agent 2. For this augmented joint state, agent 1, using GPCQ-learning, decides its action ϵ-greedily and may select an action *left* (Fig. 5 (b)). After it moves to the left, because it is no longer in an augmented joint state, it greedily selects action *right* to move back to the previous position (Fig. 5 (c)). Because the reward of selecting action *right* in a multi-agent environment is the same as that in a single-agent environment, the agent's state is not augmented, and the agent becomes trapped in repetitive movements (i.e. deadlock).



(a) Collision results in augmented joint state

(b) Different action is learned from augmented joint state

(c) Greedy action selection results in return to previous position

Fig. 5.3 Mechanism of deadlock resulting from GPCQ-learning.

5.3 Breaking a Deadlock

The deadlocks are caused by these three characteristics of GPCQ-Learning, namely, (1) actions are selected greedily in an unaugmented state, (2) the unaugmented state involved in the deadlock won't be augmented because there are no difference in instant rewards during the deadlock, (3) Q-values of the unaugmented state are not updated.

CQ-Learning does not suffer from the deadlock because actions are selected ϵ-greedily, which results in unnecessary exploration instead.

So, this thesis proposes two methods from the perspective of (2) and (3) to break a deadlock caused by GPCQ-learning. The first one is to directly detect the deadlock and augment the unaugmented state that causes the deadlock. The second one is to update Q-values of the unaugmented state that causes the deadlock to change action selection during the deadlock.

5.3.1 Augmenting the unaugmented state by detecting the deadlock

The first proposed method breaks deadlocks by directly detecting repetitive movements caused by switching between an augmented joint state and an unaugmented state and then augmenting the state of a deadlocked agent to include the state of the other agent.

The augmented states will change selection of actions, which leads to breaking the deadlock.

As shown by the underlined portions in Algorithm 5.1, the first proposed method GPCQBD (GPCQ Breaking Deadlock)-learning improves GPCQ-learning so that deadlocks are broken.

Algorithm 5.1: GPCQBD-learning algorithm for agent k

1:Train Q_k independently first, initialize Q_k^{aug} to zero and W_k=empty
2:Set t=0
3:**for** each episode **do**
4: Initialize s_k
5: **for** each step of episode **do**
6: observe local state $s_k(t)$
7: **if** $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ present in the system state s(t) **then**
8: **if** a set of $\overrightarrow{s_k}$ contains more than two $\overrightarrow{s_k}$ with $s_k(t) = s_k$ **then**
9: Select an agent l randomly from a set of $\overrightarrow{s_k}$ with $s_k(t) = s_k$
10: Select $a_k(t)$ in accordance with $Q_k^{aug}(s_k, s_l)$ ε-greedily
11: **else**
12: Select $a_k(t)$ in accordance with Q_k^{aug} ε-greedily
13: **end if**
14: **else**
15: Select $a_k(t)$ in accordance with Q_k greedily
16: **end if**
17: observe $r_k = R_k(s(t), a(t))$, s'_k from T(s(t), a(t))
18: Store <$s_k(t), a_k(t), r_k(t)$> in $W_k(s_k, a_k)$
19: **if** p-value of Student's t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
20: Store <$s_k(t), a_k(t), s_l(t), r_k(t)$> in $W_k(s_k, a_k, s_l)$ for all other agents
21: **for** all extra information s_i about another agent l Present in s(t) **do**
22: **if** p-value of Student's t-test $W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**
23: augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}
24: **end if**
25: **end for**

```
26:   end if
27:   if agent k selected a greedy action and agent k
      was/will be in the same augmented joint state at t=t-1/t+1 then
28:     augment s_k with s_l to s_k→ and add it to Q_k^aug
29:   if s_k(t) is part of s_k→ and information of s_k→ is in s(t) then
30:     if s_k' and s_l' is in part of s_k→ then
31:       Q_k^aug(s_k→, a_k) ← (1 − α_t)Q_k^aug(s_k→, a_k) + α_t[r_k + γmax_{a_k'} Q_k^aug(s_k', a_k', s_l')]
32:     else
33:       Q_k^aug(s_k→, a_k) ← (1 − α_t)Q_k^aug(s_k→, a_k) + α_t[r_k + γmax_{a_k'} Q_k(s_k', a_k')]
34:     end if
35:   else
36:     No need to update Q-value
37:   end if
38:   t = t + 1
39:  end for
40:end for
```

5.3.2 Updating Q-values of unaugmented states

The second proposed method simply updates Q-values of an unaugmented state whether the state is an interference state or not. Although the updating may break a deadlock, it may cause another issue. If an agent coincidentally collides with another agent at a specific state, the Q-value of the action it took at the state is underestimated due to the instant penalty of -10 and the agent tends to take other actions at the state without an appropriate reason. The pros and cons of applying the method must be evaluated.

The second proposed method GPCQwU (GPCQ with Updating)-learning is described in detail in Algorithm 5.2.

Algorithm 5.2: GPCQwU-learning algorithm for agent k

```
1:Train Q_k independently first, initialize Q_k^aug to zero and W_k=empty
2:Set t=0
```

3:**for** each episode **do**

4: Initialize s_k

5: **for** each step of episode **do**

6: observe local state $s_k(t)$

7: **if** $s_k(t)$ is part of a $\overrightarrow{s_k}$ and the info of $\overrightarrow{s_k}$ present in the system state s(t) **then**

8: **if** a set of $\overrightarrow{s_k}$ contains more than two $\overrightarrow{s_k}$ with $s_k(t) = s_k$ **then**

9: Select an agent l randomly from a set of $\overrightarrow{s_k}$ with $s_k(t) = s_k$

10: Select $a_k(t)$ in accordance with $Q_k^{aug}(s_k, s_l)$ ε-greedily

11: **else**

12: Select $a_k(t)$ in accordance with Q_k^{aug} ε-greedily

13: **end if**

14: **else**

15: Select $a_k(t)$ in accordance with Q_k greedily

16: **end if**

17: observe $r_k = R_k(s(t), a(t))$, s'_k from $\mathrm{T}(\mathrm{s}(\mathrm{t}), \mathrm{a}(\mathrm{t}))$

18: Store $<s_k(t), a_k(t), r_k(t)>$ in $W_k(s_k, a_k)$

19: **if** p-value of Student's t-test $(W_k(s_k, a_k), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**

20: Store $<s_k(t), a_k(t), s_l(t), r_k(t)>$ in $W_k(s_k, a_k, s_l)$ for all other agents

21: **for** all extra information s_i about another agent l Present in s(t) **do**

22: **if** p-value of Student's t-test $W_k(s_k, a_k, s_l), \mathrm{E}(R_k(s_k, a_k)) < p_{th}$ **then**

23: augment s_k with s_l to $\overrightarrow{s_k}$ and add it to Q_k^{aug}

24: **end if**

25: **end for**

26: **end if**

27: **if** $s_k(t)$ is part of $\overrightarrow{s_k}$ and information of $\overrightarrow{s_k}$ is in s(t) **then**

28: **if** s'_k and s'_l is in part of $\overrightarrow{s_k}$ **then**

29: $Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1 - \alpha_t) Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t [r_k + \gamma max_{\alpha'_k} Q_k^{aug}(s'_k, a'_k, s'_l)]$

30: **else**

31: $Q_k^{aug}(\overrightarrow{s_k}, a_k) \leftarrow (1 - \alpha_t) Q_k^{aug}(\overrightarrow{s_k}, a_k) + \alpha_t [r_k + \gamma max_{\alpha'_k} Q_k(s'_k, a'_k)]$

32: **end if**

33: **else**

34: $Q_k(s_k, a_k) \leftarrow (1 - \alpha_t) Q_k(s_k, a_k) + \alpha_t [r_k + \gamma max_{\alpha'_k} Q_k(s'_k, a'_k)]$

35: **end if**

36: $t = t + 1$

```
37:  end for
38:end for
```

5.4 Evaluation

5.4.1 Pursuit games

The proposed learning methods are evaluated in comparison with existing methods: independent learning, JSQ-learning, JSAQ-learning, CQ-learning, and GPCQ-learning. The number of episodes was set to 20,000 for independent learning, JSQ-learning, and JSAQ-learning and 10,000 for CQ-, GPCQ-, and improved GPCQ-learning because CQ-learning and its extensions require prelearning (in this case, 10,000 episodes) in a single-agent environment. In the prelearning, the initial positions of the agents and the target are randomly selected, and ϵ was set to 0.8 to ensure that the agents could sufficiently explore the environment.

The seven initial agent/target-position patterns shown in Fig. 5.2 were used for our evaluation. The length of the window used to calculate the distribution of immediate rewards was set to 20. The threshold of the Student's t-test, p_{th}, was set to 0.01, as was done by Hauwere et al. [Hauwere (2010, 2011)].

Table 5.2 shows the average number of steps to finish and the standard deviation. The smallest number of steps and the deviation are depicted with hatches.

For patterns 1-3, the two proposed methods, as well as GPCQ-learning, found the optimal paths because there were no interferences between the agents if the agents greedily selected their actions.

A slight improvement was obtained for pattern 4 by GPCQBD-learning method. GPCQBD-learning method substantially outperformed GPCQ-learning for patterns 5-7 while the performance of GPCQ-learning was worst because of deadlocks.

The agents trained using GPCQwU-learning had the best performance in all the patterns. It must be noticed that the deviation of steps to finish has been converged to zero in all the patterns with GPCQwU-learning; indicating GPCQwU-learning tends to lead agents to a local minimum of their action policies. In fact, for patterns 4-7, the agents

trained using GPCQwU-learning failed to find the shortest steps to their goals while they succeeded to avoid collisions with other agents.

Table 5.2 Evaluation of average number of steps to finish in pursuit games.

Patterns	1		2		3		4		5		6		7	
Minimum Steps	4		4		4		4		3		4		5	
Methods	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.
Independent	5.07	1.56	5.65	1.76	5.02	1.54	6.77	2.56	5.11	1.72	6.40	5.07	8.32	7.27
JSQ	5.23	2.35	5.38	2.82	6.87	8.14	6.69	4.12	5.22	2.47	6.59	4.91	11.5	11.8
JSAQ	6.91	9.65	7.11	10.5	108	110	7.73	14.2	7.47	10.7	8.54	31.2	141	119
CQ	5.15	1.73	6.45	8.88	6.26	13.7	5.87	1.57	18.8	28.2	178	291	30.4	54.0
GCQ	4.00	0.00	4.00	0.00	4.00	0.00	5.12	0.482	–	–	–	–	–	–
PCQ	5.17	1.96	6.69	10.1	6.00	18.5	5.94	1.63	26.4	30.5	56.6	57.8	43.5	109
GPCQ	4.00	0.00	4.00	0.00	4.00	0.00	5.14	0.518	–	–	–	–	–	–
GPCQBD	4.00	0.00	4.00	0.00	4.00	0.00	5.09	0.443	7.64	2.09	5.50	1.05	8.98	2.53
GPCQwU	4.00	0.00	4.00	0.00	4.00	0.00	5.00	0.00	4.00	0.00	5.00	0.00	6.00	0.00

Figure 5.4 shows learning curves of each method, CQ-learning and its enhancements, in pursuit games.

In pattern 1-3, the agents learned using CQ- and PCQ-learning had poorer performance due to ϵ-greedy action selection while other agents perform optimally because there was no collision if they took greedy actions.

In pattern 4, all the method failed to find the optimal steps to capture the target because agent 2 tended to go to the upside of the target and agent 1 had to go to the left of right of the target. GPCQwU-learning converged to the local minimum, i.e. 5 steps to finish.

In pattern 5-7, the agents learned using GCQ- and GPCQ-learning rarely finish the games due to greedy action selection that caused deadlocks. Any method managed to find the optimal steps to capture the target. The agent learned using GPCQwU-learning had the best performance.

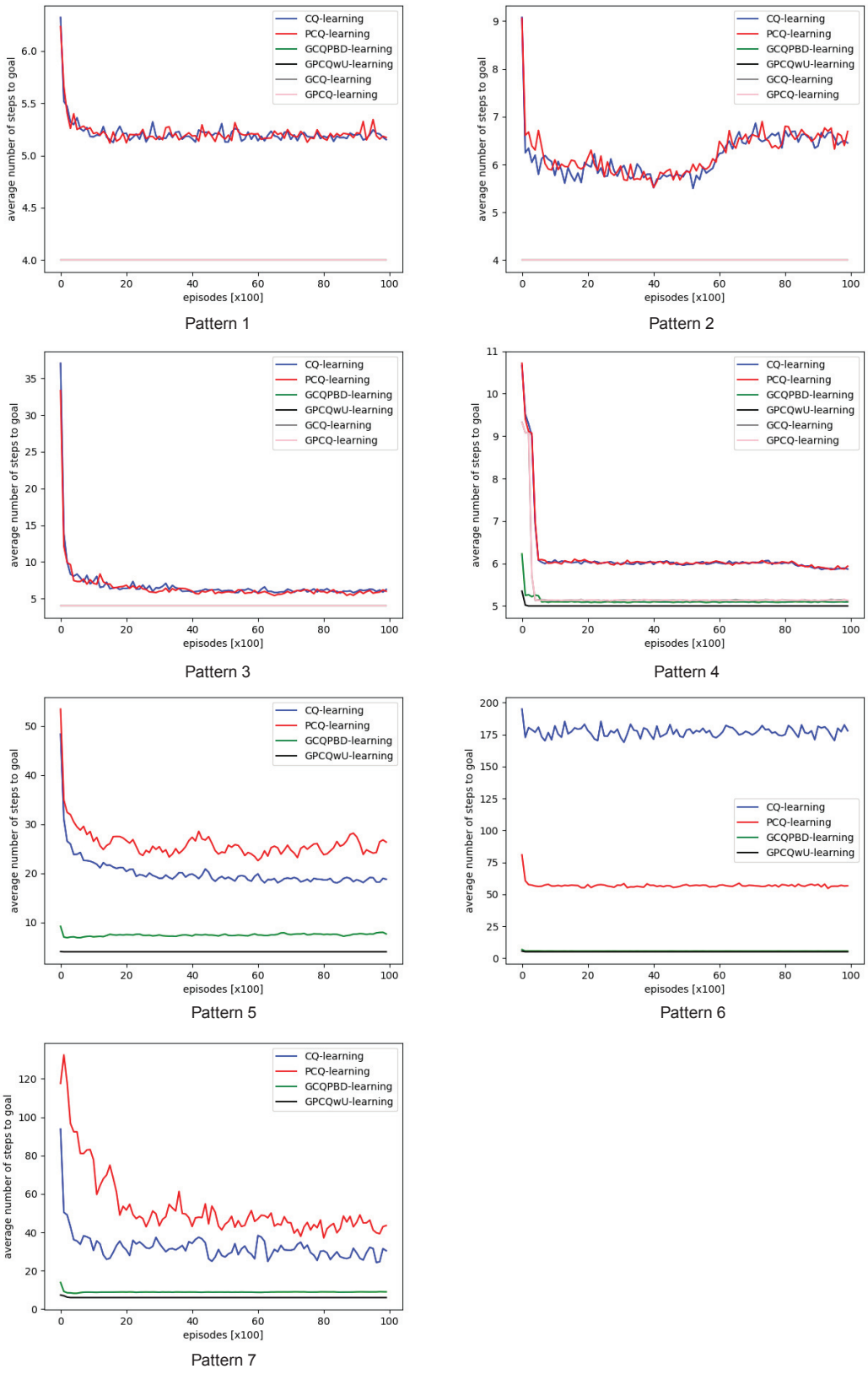


Fig. 5.4 Learning curves of each method in pursuit games.

5.4.2 Maze games

This section also evaluates each method in maze games used in Chapter 4. The agents learned using GPCQwU-learning finally found the shortest path to goal in ISR and outperformed other methods in TunnelToGoal3. It must be noted that the agents learned using GPCQwU-learning converged to the local minimum, which took 24 steps.

Table 5.3 Evaluation of average number of steps to goal in maze games.

	Tunnel2Goal		ISR		CIT		CMU		TunnelToGoal3	
	mean	std	mean	std	mean	std	mean	std	mean	std
Min Steps	11		6		13		33		12	
CQ	13.5	3.84	31.6	79.9	25.7	26.9	41.7	8.58	23.8	19.2
GCQ	13.2	4.87	51.2	229	32.9	133	33.8	5.01	28.2	16.7
PCQ	12.8	1.59	8.64	5.87	19.1	15.2	39.4	4.56	23.1	17.7
GPCQ	11.3	0.616	7.42	1.68	15.6	7.68	33.3	2.94	30.9	22.6
GPCQBD	11.4	0.720	7.00	1.13	16.8	14.73	33.1	0.37	21.8	11.8
GPCQwU	11.4	1.61	6.03	0.17	24.0	0.00	33.0	0.00	14.5	0.883

Fig. 5.5 shows how the agent learned using GPCQwU-learning failed into the local minimum. In the single-agent environment, each agent learned to take the shortest path to goal, resulting in collisions in the multi-agent environment. The number of the shortest steps for the ISR game is 13 if the agent 1 takes the shortest path and the agent 2 waits at the location marked with a red circle while the agent 1 passes.

In all the method except for GPCQwU-learning, because Q-values of unaugmented states are fixed during learning process, the agents tend to take the shortest path to goal and collide with another agent, resulting in augmentation of the interference states. So, they have chances to learn how to coordinate by updating the Q-values of the augmented states.

In GPCQwU-learning, once a collision occurs if the agents take the shortest path, they tend to avoid the shortest path. If one of the agents finds a detour route, the action to the route is reinforced and they loose chances to learn how to coordinate. For example, both agent 1 and agent 2 first take the shortest path and collide with each other at the location marked with an orange cross. Because CQ-learning and its enhancements need multiple times of collisions, i.e., 20 times in this setting, to detect a difference of the instant reward

between in the single-agent environment and in the multi-agent environment, Q-values of the greedy action learned in the single-agent environment is underestimated by the collisions. If agent 2 finds a detour route depicted with a red line before augmentation of the interference states, the actions to the detour route are reinforced. Once the greedy actions from S_2 to the location marked with a face, i.e., from the location to the G_2 actions that lead to the red line are already greedy, change to the actions that lead to the detour route, agent 2 never try to find the optimal path because augmentation will never occur.

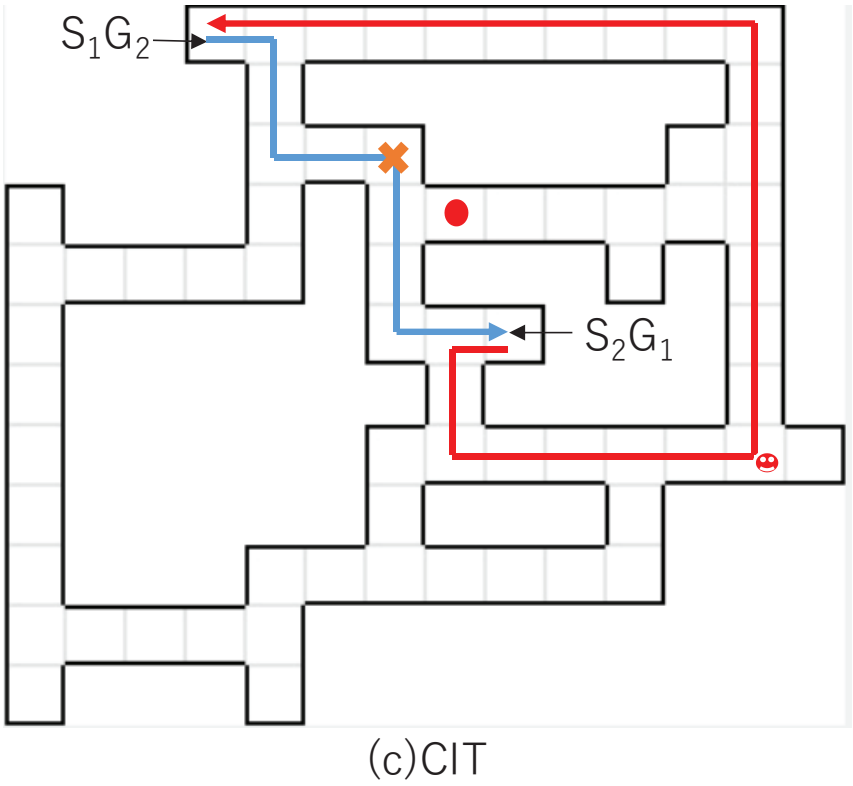


Fig. 5.5 An example of converging to a local minimum.

Figure 5.6 shows the learning curves of each learning method in five maze games.

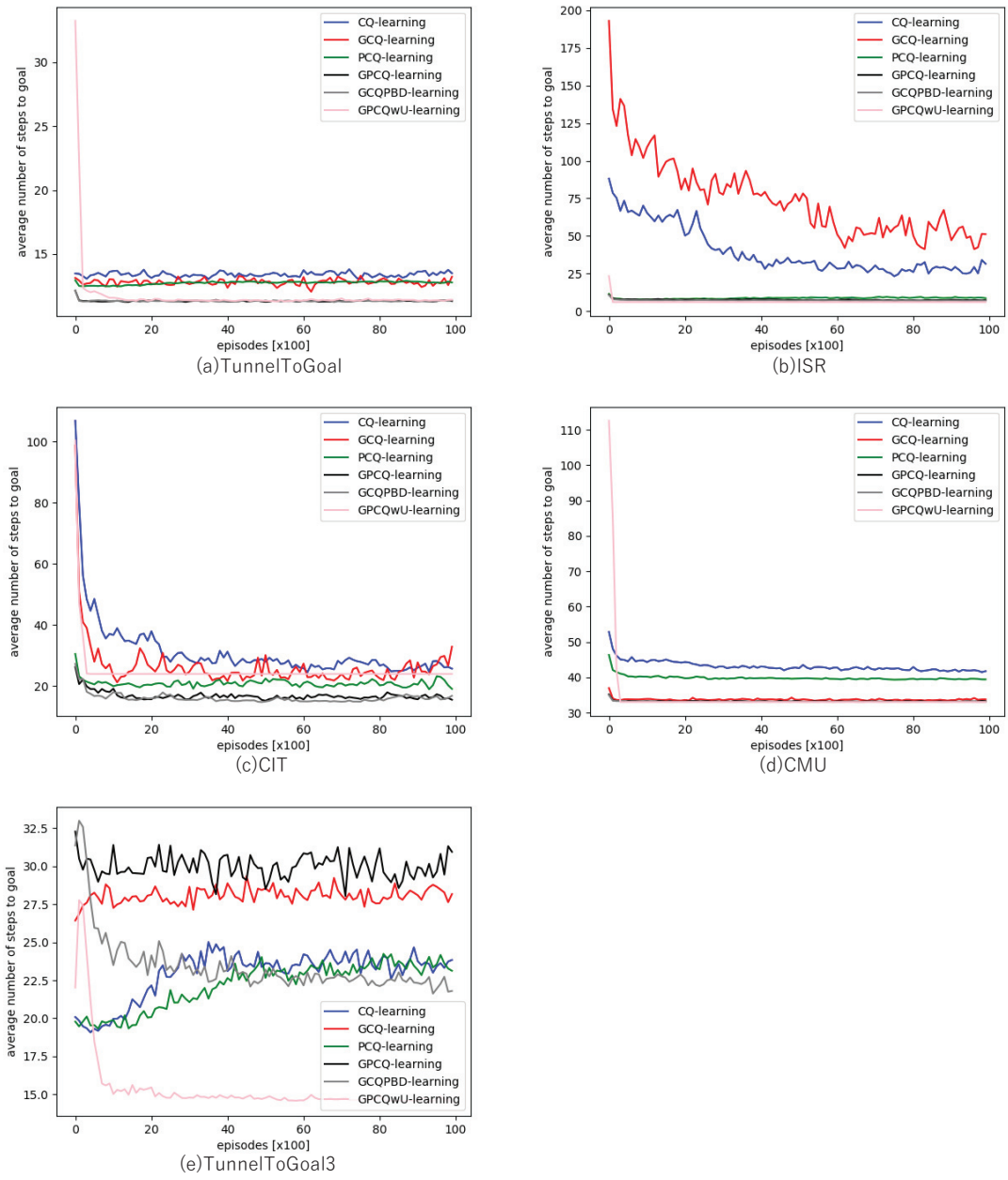


Fig. 5.6 Learning curves of each method in maze games.

5.5 Summary

This chapter points out that GPCQ-learning sometimes causes a deadlock because an oscillation between an unaugmented state and an augmented state makes no difference in instant rewards between in the single-agent environment and in the multi-agent

environment.

This thesis proposes two methods to solve the deadlock. The first one, GPCQBD-learning, directly detects the deadlock and augments the unaugmented state involving in the deadlock. The second one, GPCQwU-learning, updates Q-values of unaugmented states.

Evaluation using pursuit games and maze games demonstrated that both methods, especially GPCQwU-learning, improve the performance of GPCQ-learning while GPCQwU-learning has chances to fall in a local minimum.

Chapter 6 Conclusion

6.1 Contributions

- This thesis pointed out that CQ-learning, which is one of reinforcement learning method for Dec-SIMDP, has at least six issues to be improved; namely how prelearning should be conducted, unnecessary exploration by ϵ-greedily action selection, optimistic Q-value updating, which Q-values should be used if more than two agents involve in an interference, a problem in a multi-agent environment must be manually converted multiple problems in a single-agent environment, and (6) it only detects a difference of immediate rewards to identify interfered states.

- This thesis presented four approaches to solve the issues. The first approach for prelearning is to set ϵ value of ϵ-greedily action selection in a single-agent environment to 0.8 to ensure that an agent can explore all state-action combination enough. The second approach for avoiding unnecessary exploration is making an agent select its action greedily if it is in an unaugmented state exploiting knowledge learned in a single-agent environment. The third approach for avoiding optimistic Q-value updating is to change Q-value updating equation based on whether an agent is still in an interference state after taking previous action. The last approach for dealing with interference among more than two agents is randomly selecting one agent from agents that are in the interference state.

- Evaluation using five maze games demonstrates that if both greedy action selection and changing Q-value updating equation based on whether an agent is still in an interference state after taking previous action are applied, we call the learning method GPCQ-learning, the performance of CQ-learning is improved substantially.

- This thesis pointed out that in some pursuit games GPCQ-learning fell into a deadlock due to greedy action selection at an unaugmented state and failing to detect the deadlock

because there was no difference of instant reward between in a single-agent environment and in a multi-agent environment.

- This thesis proposed two approaches to break a deadlock caused by GPCQ-learning. The first approach is directly detecting the deadlock and augmenting the unaugmented state. The second approach is updating Q-values of unaugmented states as well as augmented states.

- Evaluation using seven patterns of pursuit games and five maze games demonstrates that the two proposed approaches improved the performance of GPCQ-learning by breaking a deadlock.

6.2 Future Work

- A conversion of a problem in a multi-agent environment to multiple problems in a single-agent environment is manually designed. Some guidelines or theories are required.

- The number of episodes in prelearning is manually designed. Some criteria are necessary to automatically decide it and maintain the performance.

- More suitable way should be used to select one of agents when more than two agents are in an interference state.

- As mentioned in Chapter 5, GPCQwU-learning has a problem of falling into a local minimum while it performs best in most of games.

- CQ-learning and its enhancements can be applied only to problems in which an interference causes a difference in instant rewards. For example, if a penalty of a collision is set to zero in pursuit games and maze games, agents in the environment can not detect

any difference in instant rewards. The only difference is the cumulative reward to the end of the games, i.e., number of steps.

- The implementation of the proposed methods in this thesis is based on simple table lookups of Q-table for unaugmented states and a dictionary of augmented states. It is necessary to implement them using deep neural networks if the state-action space is huge in a real problem. When an augmented state is generated, values of $Q_k(s_k)$ are copied to $Q_k^{aug}(s_k, s_l)$ for each s_l in our implementation. The sam mechanism of copies must be achieved between deep neural networks approximating $Q_k(s_k)$ and those approximating $Q_k^{aug}(s_k, s_l)$.

Appendix

Appendix A: Class List and Diagram

Here is a list of Python classes used in this thesis.

Table B.1 Class list.

#	Class Name	Role
1	Agent	Base class of an agent that holds basic information like, the number of states, the number of actions, information of its environment, and Q-values.
2	JSQLearner	Joint-state Q-learning.
3	JSAQLearner	Joint-state-action Q-learning.
4	CQLearner	Class of CQ-learning that holds information about interference states and augmented states.
5	GCQLearner	CQ-learning with greedy action selection.
6	PCQLearner	CQ-learning with pessimistic Q-value updating.
7	GPCQLearner	CQ-learning with both greedy action selection and pessimistic Q-value updating.
8	GPCQBDLearner	GPCQ-learning with detecting a deadlock and augmenting the involving state.
9	GPCQwULearner	GPCQ-learning with updating unaugmented states.
10	Qtable	A table that holds Q-values of each state.
11	AugmentedQ	A set of Q-values of each augmented state.
12	Environment	Base class of an environment that provides basic functions, like env_init(), env_update(), observe(), etc.
13	MultiMaze	Environment of maze games
14	MultiPursuit	Environment of pursuit games.
15	MultiAgentRL	Operating common multi-agent reinforcement algorithm.
16	Maze	Base class of maze games.

17	TunnelToGoal	Class of the TunnelToGoal game.
18	ISR	Class of the ISR game.
19	CIT	Class of the CIT game.
20	CMU	Class of the CMU game.
21	TunnelToGoal3	Class of the TunnelToGoal3 game.
22	EvalMaze	Operating evaluation of each method in maze games.
23	EvalPursuit	Operating evaluation of each method in pursuit games.

The class diagram of above classes is shown in Fig. A.1.

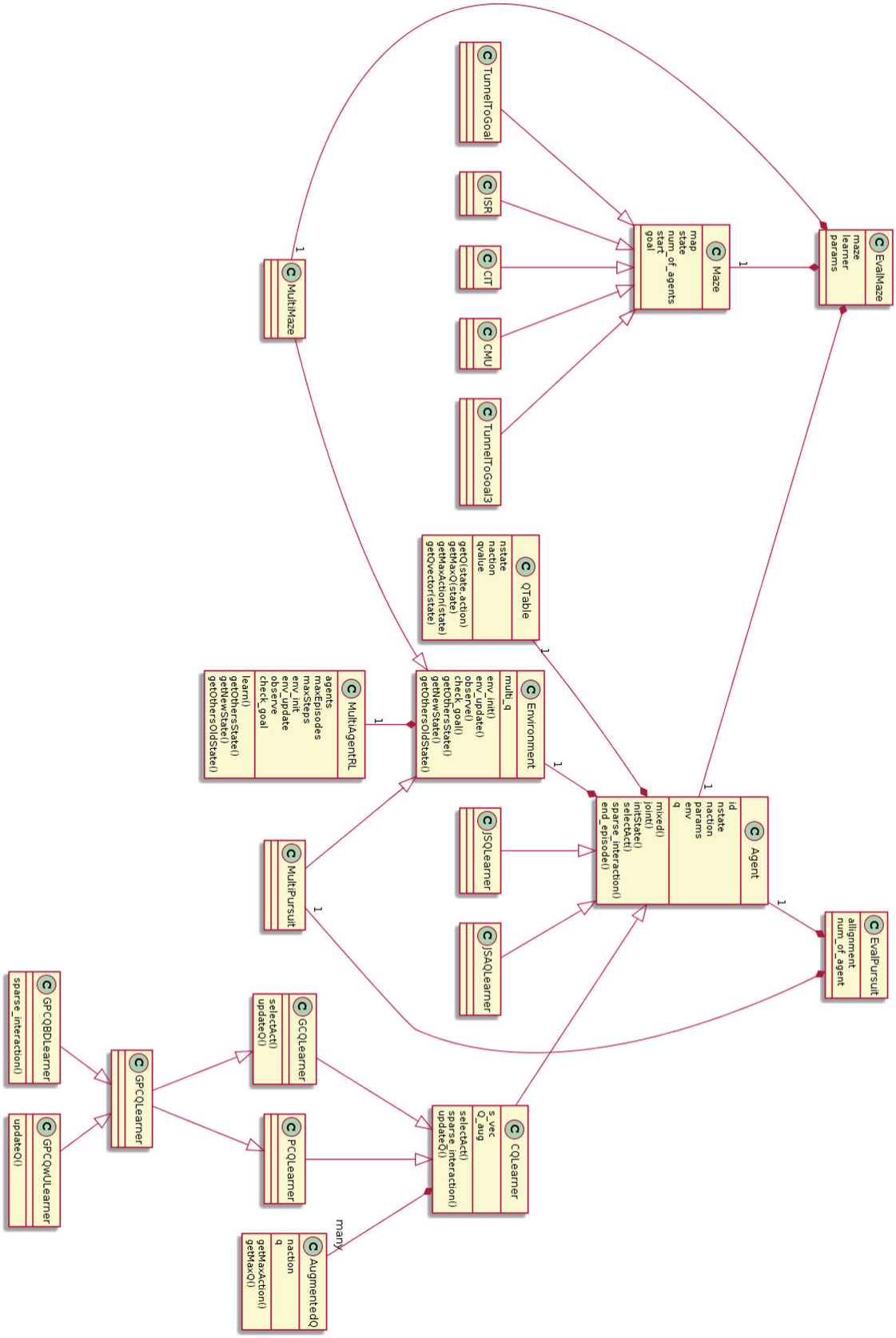


Fig A.1 Class Diagram

Bibliography

1. [Arai & Ishigaki (2009)] S. Arai and Y. Ishigaki: Information Theoretic Approach for Measuring Interaction in Multiagent Domain, Journal of Advanced Computational Intelligence and Intelligent Informatics Vol. 13 No. 6, pp. 649-657 (2009).
2. [Arai & Xu (2016)] S. Arai and H. Xu: Faster convergence to cooperative policy by autonomous detection of interference states in multiagent reinforcement learning, In Proceedings of PRICAI 2016, pp. 16-19 (2016).
3. [Aras et. al (2004)] R. Aras, A. Dutech, and F. Charpillet: Cooperation through communication in decentralized Markov games, In Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications (2004).
4. [Bellman (1957)] R. E. Bellman: Dynamic Programming, Princeton University Press, Princeton (1957).
5. [Bernstein et al. (2002)] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein: The Complexity of Decentralized Control of Markov Decision Processes, Mathematics of Operations Research, Vol. 27, pp. 819-840 (2002).
6. [Bloembergen et al. (2015)] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers: Evolutionary dynamics of multi-gent learning: a survey, Journal of Artificial Intelligence Research, Vol. 53, Issue 1, pp. 659–697 (2015).
7. [Boutilier (1996)] C. Boutilier: Planning, learning and coordination in multiagent decision processes, In Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge, pp. 195–210 (1996).
8. [Buşoniu et al. (2008)] L. Buşoniu, R. Babuška, and B. De Schutter: A Comprehensive Survey of Multi-Agent Reinforcement Learning, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews Vol. 38 No. 2, pp. 156-172 (2008).
9. [Buşoniu et al. (2010)] L. Buşoniu, R. Babuška, and B. De Schutter: Multi-agent reinforcement learning: An overview, Chapter 7 in Innovations in Multi-Agent Systems and Applications – 1 (D. Srinivasan and L.C. Jain, eds.), Vol. 310 of Studies in Computational Intelligence, Berlin, Germany: Springer, pp. 183–221 (2010).
10. [Claus & Boutilier (1998)] C. Claus and C. Boutilier: The dynamics of reinforcement learning in cooperative multiagent systems, In Proceedings of the 15th National Conference on Artificial Intelligence, pp. 746–752 (1998).
11. [Guestrin et al. (2002)a] C. Guestrin, M. Lagoudakis, and R. Parr: Coordinated reinforcement learning, In Proceedings of the 19th International Conference on Machine Learning, pp. 227-234 (2002).
12. [Guestrin et al. (2002)b] C. Guestrin, S. Venkataraman, and D. Koller: Context-specific multiagent coordination and planning with factored MDPs, In Proceedings of the 19th National Conference on Artificial Intelligence, pp. 253-259 (2002).
13. [Hauwere et al. (2009)] Y. Hauwere, P. Vrancx, and A. Nowé: Learning what to observe in multi-agent systems, In Proceedings of the 21st Benelux Conference on Artificial Intelligence, pp. 83-90 (2009)
14. [Hauwere et al. (2010)] Y. Hauwere, P. Vrancx, and A. Nowé: Learning multi-agent state space representations, In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 715–722 (2010).

15. [Hauwere (2011)] Y. Hauwere: *Sparse Interactions in Multi-Agent Reinforcement Learning*, Ph.D. Thesis, Vrije Universiteit Brussel (2011).
16. [Hauwere (2012)] Y. Hauwere, S. Devlin, D. Kudenko, and A. Nowé: Context-Sensitive Reward Shaping for Sparse Interaction Multi-Agent Systems, Knowledge Engineering Review Vol. 31, pp. 59-76 (2016)
17. [Hernandez et al. (2017)] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote: A survey of learning in multiagent environments: dealing with non-stationarity, arXiv:1707.09184v1 (2017).
18. [Hu and Gao (2015)] Y. Hu , Y. Gao, and B. An: Multiagent Reinforcement Learning with Unshared Value Functions, IEEE Transaction on Cybernetics Vol. 45, No. 4, pp. 647-662 (2015).
19. [Kaelbling et al. (1996)] L. P. Kaelbling, M. L. Littman, and A. W. Moore: Reinforcement Learning: A survey, Journal of Artificial Intelligence Research Vol. 4, Issue 1, pp. 237-285 (1996).
20. [Kar et al. (2013)] S. Kar, J. M. F. Noura, and H. V. Poor: QD-learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus + Innovations, IEEE Transaction on Signal Processing Vol. 61 No. 7, pp. 1848-1862 (2013).
21. [Kok & Vlassis (2004)a] J. R. Kok and N. Vlassis: Sparse cooperative Q-learning, In Proceedings of 21st International Conference on Machine learning, pp. 61-68 (2004).
22. [Kok & Vlassis (2004)b] J. R. Kok and N. Vlassis: Sparse tabular multiagent Q-learning, In Proceedings of the 13th Annual Machine Learning Conference of Belgium and the Netherlands, pp. 65-71 (2004).
23. [Kok et al. (2005)] J. R. Kok, P. J. Hoen, B. Bakker, and N. Vlassis: Utile coordination: learning interdependencies among cooperative agents, In Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 29-36 (2005).
24. [Kok & Vlassis (2006)] J. R. Kok and N. Vlassis: Collaborative Multiagent Reinforcement Learning by Payoff Propagation, Journal of Machine Learning Research, Vol. 7, pp. 1789-1828 (2006).
25. [Kujirai & Yokota (2018)] T. Kujirai and T. Yokota: Greedy action selection and pessimistic Q-value updates in cooperative Q-learning, In Proceedings of the SICE Annual Conference, pp. 821-826 (2018).
26. [Kujirai & Yokota (2019)] T. Kujirai and T. Yokota: Greedy action selection and pessimistic Q-value updating in multi-agent reinforcement learning with sparse interaction, SICE Journal of Control, Measurement, and System Integration, Vol. 12, No. 3, pp. 76-84 (2019).
27. [Lauer & Riedmiller (2000)] M. Lauer and M. Riedmiller: An algorithm for distributed reinforcement learning in cooperative multi-agent systems, In Proceedings of the 17th International Conference on Machine Learning, pp. 535–542 (2000).
28. [Melo & Veloso (2009)] F. Melo and M. Veloso: Learning of coordination: exploiting sparse interactions in multiagent systems, In Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, pp. 773–780 (2009).
29. [Melo & Veloso (2010)] F. Melo and M. Veloso: Local Multiagent Coordination in Decentralized MDPs with Sparse Interactions, CMU-CS-10-133, School of Computer Science, Carnegie Mellon University, Pittsburgh (2010).

30. [Melo & Veloso (2011)] F. Melo and M. Veloso: Decentralized MDPs with sparse interactions, Artificial Intelligence, Vol. 175, Issue. 11, pp. 1757–1789 (2011).
31. [Puterman (1994)] M. L. Puterman: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, New York (1994).
32. [Rummery & Niranjan (1994)] G. A. Rummery and M. Niranjan: On-line Q-learning using connectionist systems, Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department (1994).
33. [Russell & Norvig (2003)] S. Russell and P. Norvig: *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 2nd edition (2003).
34. [Scharpff et al. (2016)] J. Scharpff, D. M. Roijers, F. A. Oliehoek, M. T. J. Spaan, and M. M. de Weerdt: Solving transition-independent multi-agent MDPs with sparse interactions, In Proceedings of the 30th AAAI Conference on Artificial Intelligence, pp. 3174-3180 (2016)
35. [Sen et al. (1994)] S. Sen, M. Sekaran, and J Hale: Learning to coordinate without sharing information, In Proceedings of the 12th National Conference on Artificial Intelligence, pp. 426–431 (1994).
36. [Spaan & Melo (2008)] M. T. J. Spaan and F. S. Melo: Interaction-driven Markov games for decentralized multiagent planning under uncertainty, In Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 525-532 (2008).
37. [Stone & Sutton (2001)] P. Stone and R. Sutton: Scaling reinforcement learning toward Robocup soccer, In Proceedings of the 18th International Conference on Machine Learning, pp. 537-544 (2001).
38. [Sutton (1984)] R. S. Sutton: *Temporal Credit Assignment in Reinforcement Learning*, Ph.D. thesis, University of Massachusetts, Amherst, MA (1984).
39. [Sutton (1988)] R. S. Sutton: Learning to predict by the method of temporal differences, Machine Learning, Vol.3 Issues 1, pp. 9-44 (1988).
40. [Tan (1993)] M. Tan: Multi-agent reinforcement learning: independent vs. cooperative agents, In Proceedings of the 10th International Conference on Machine Learning, pp. 330–337 (1993).
41. [Tsitsiklis (1994)] J. Tsitsiklis: Asynchronous stochastic approximation and Q-learning, Journal of Machine Learning Vol. 16 No. 3. Pp. 185-202 (1994).
42. [Vlassis (2007)] N. Vlassis: A concise introduction to multiagent systems and distributed artificial intelligence, Synthesis Lectures on Artificial Intelligence and Machine Learning (2007).
43. [Watkins (1989)] C. J. C. H. Watkins: *Learning from Delayed Rewards*, Ph.D. Thesis, Cambridge University (1989).
44. [Watkins & Dayan (1992)]C. J. C. H. Watkins and P. Dayan: Q-learning, Machine Learning, Vol. 8, Issue 3-4, pp. 279–292 (1992).
45. [Yu et al. (2014)] C. Yu, M. Zhang, and F. Ren: Coordinated Learning by Exploiting Sparse Interaction in Multiagent Systems, Concurrency Computation.: Practice and Experience Vol. 26 No. 1, pp. 51-70 (2014).
46. [Yu et al. (2015)] C. Yu, M. Zhang, F. Ren, and G. Tan: Multiagent Learning of Coordination in Loosely Coupled Multiagent Systems, IEEE Transaction on Cybernetics Vol. 45 No. 12, pp. 2853-2867 (2015).

47. [Zhang & Lesser (2013)] C. Zhang and V. Lesser: Coordinating multi-agent reinforcement learning with limited communication, In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems, pp. 1101-1108 (2013).
48. [Zhou et al. (2017)] L.Zhou, P. Yang, and C. Chen: Multi-agent reinforcement learning with sparse interactions by negotiation and knowledge transfer, IEEE Transaction on Cybernetics, Vol. 47, No. 5, pp. 1238-1250 (2017).
49. [市川 & 高玉 (2012)] 市川 嘉裕・高玉 圭樹, 学習進度に基づくマルチエージェントQ学習における競合回避, 計測自動制御学会論文集 Vol. 48 No. 11, pp. 764-772 (2012)
50. [齋藤 & 小林 (2014)] 斎藤 碧・小林 一郎, 強化学習における効率的な転移学習適用に関する一考察, 人工知能学会全国大会論文集（2014）.
51. [篠塚 & 若林 (2014)] 篠塚 敬介・若林 啓, 異なる環境に適用可能な知識を考慮した強化学習手法の構築, DEIM Forum (2014).
52. [山田 & 高野 (2013)] 山田 和明・高野 慧, マルチエージェントシステムのための信頼度を用いた強化学習 - 相補ゲームにおける競合回避行動の獲得 -, 計測自動制御学会論文集 Vol. 49 No.1, pp. 39-47 (2013).
53. [山分 et al. (2013)] 山分 翔太・黒江 康明・飯間 等, マルチエージェントタスクに対する群強化学習法 – ジレンマ問題の解法 -, 計測自動制御学会論文集 Vol. 49 No. 3, pp. 370-377 (2013)