Machine Learning, 59, 31–54, 2005

A Reinforcement Learning Scheme for a Partially-Observable Multi-Agent Game

SHIN ISHII ishii@is.naist.jp
Nara Institute of Science and Technology, CREST, Japan Science and Technology Agency, 8916-5 Takayama, Ikoma, 630-0192 Japan

HAJIME FUJITA
MASAOKI MITSUTAKE
Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, 630-0192 Japan

TATSUYA YAMAZAKI
National Institute of Information and Communications Technology, 3-5 Hikaridai, Seika, Kyoto, 619-0289 Japan

JUN MATSUDA
Osaka Gakuin University, 2-36-1 Kishibeminami, Suita, 564–8511 Japan

YOICHIRO MATSUNO
Ricoh Co. Ltd. 1-1-17 Koishikawa, Tokyo, 112-0002 Japan

Editor: Risto Miikkulainen

Abstract. We formulate an automatic strategy acquisition problem for the multi-agent card game “Hearts” as a reinforcement learning problem. The problem can approximately be dealt with in the framework of a partially observable Markov decision process (POMDP) for a single-agent system. Hearts is an example of imperfect information games, which are more difficult to deal with than perfect information games. A POMDP is a decision problem that includes a process for estimating unobservable state variables. By regarding missing information as unobservable state variables, an imperfect information game can be formulated as a POMDP. However, the game of Hearts is a realistic problem that has a huge number of possible states, even when it is approximated as a single-agent system. Therefore, further approximation is necessary to make the strategy acquisition problem tractable. This article presents an approximation method based on estimating unobservable state variables and predicting the actions of the other agents. Simulation results show that our reinforcement learning method is applicable to such a difficult multi-agent problem.

Keywords: reinforcement learning, POMDP, multi-agent system, card game, model-based

1. Introduction

Many card games are imperfect information games; for each game player, there are unobservable state variables, e.g., cards in another player’s hand or undealt cards. Since card games are well-defined as multi-agent systems, strategy acquisition problems for them have been widely studied. However, the existing algorithms have not achieved the level of human experts (Ginsberg, 2001), although some algorithms for perfect information games like the game “Backgammon” can beat human champions (Tesauro, 1994). In order to deal with imperfect information games, it is important to estimate missing information (Ginsberg, 2001).

A decision making problem or an optimal control problem in a stochastic but stationary environment is often formulated as a Markov decision process (MDP). On the other hand, if the information in the environment is partially unobservable, the problem can be formulated as a partially observable Markov decision process (POMDP). By regarding missing information as unobservable part of the environment, an imperfect information game is formulated as a POMDP.

In many card games, coordination and competition among the players occur. Such a situation is referred to as a multi-agent system. A decision making problem or an optimal control problem in a multi-agent system has a high degree of difficulty due to interactions among the agents. Reinforcement learning (RL) (Sutton & Barto, 1998), which is a machine learning framework based on trial and error, has often been applied to problems within multi-agent systems (Crites, 1996; Crites & Barto, 1996; Littman, 1994; Hu & Wellman, 1998; Nagayuki, Ishii, & Doya, 2000; Salustowicz, Wiering, & Schmidhuber, 1998; Sandholm & Crites, 1995; Sen, Sekaran, & Hale, 1994; Tan, 1993), and has obtained successful results.

This article in particular deals with the card game “Hearts”, which is an n-player ($n > 2$) non-cooperative finite-state zero-sum imperfect-information game, and presents an automatic strategy-acquisition scheme for the game. By approximately assuming that there is a single learning agent, the environment can be regarded as stationary for the agent. The strategy acquisition problem can then be formulated as a POMDP, and the problem is solved by an RL method. Our RL method copes with the partial observability by estimating the card distribution in the other agents’ hands and by predicting the actions of the other agents. After that, we try to apply our POMDP-RL method to a multi-agent problem, namely, an environment that has several agents that learn concurrently.

In a POMDP, the state transition for an observable part of the environment, i.e., observable state variables, does not necessarily have a Markov property. A POMDP can be transformed into an MDP whose state space consists of belief states. A belief state is typically the probability distribution of possible states. After each state transition for the observable state variables occurs, the belief state maintains the probability of the unobservable part of the environment; namely, the belief state is estimated using the observations of actual state transition events. If the correct model of the environmental dynamics is available, the optimal control (i.e., “policy”) for a POMDP is obtained based on a dynamic programming (DP) approach (Kaelbling, Littman, & Cassandra, 1998). The agent does not have *a priori* knowledge of the environmental dynamics in usual RL problems, hence, it is important for a POMDP-RL method to be able to estimate the environmental model.

In the game Hearts, the environmental model (state transition) depends on the cards held by opponent agents and the strategies (actions) of the opponent agents. Therefore, a good estimation for the state transition probability needs to approximate the card distribution and the action prediction for the opponent agents. This approximation problem is difficult in comparison to those in the existing POMDP-RL studies or the existing multi-agents studies; namely, the learning of the game Hearts is a realistic problem.

The game Hearts belongs to a class of perfect recall games, which is a subclass of the class of imperfect information games. A perfect recall game assumes that an agent remembers the complete history of state transitions in the past. Since obtaining the optimal strategy in a perfect recall game with n players ($n \geq 2$) is known to be *NP*-hard (Blair, Mutchler, & Lent, 1995), we need some approximation to solve its optimal control problem within a reasonable computational time.

Our RL scheme is derived based on a formulation of a single-agent POMDP. Then, the most parts of this article assume that there is only one learning agent in the environment. If there are two or more learning agents, the environment does not have a Markov property, and it cannot be formulated rigorously as a POMDP. However, we assume that the environment is approximately stationary for every learning agent. In order to let this assumption be valid, the learning of the environmental model is faster than the changes of the other agents. If such fast learning is realized, by using efficient function approximators for example, our method can be applied to concurrent learning situations of multiple agents. Since the strategy of an opponent agent may differ from that of another opponent agent, we individually prepare a function approximator representing the policy of each opponent agent. If the strategy of one agent changes, it is enough for a single function approximator to adapt to the change independently; this provides us with an efficient and robust learning scheme for multi-agent problems.

2. Partially-observable Markov decision process

A POMDP is defined by state transition probability $P(s_{t+1} \mid s_t, a_t)$, observation probability $P(x_t \mid s_t)$, and reward function $r_t \equiv R(s_t, a_t)$ (Kaelbling, Littman, & Cassandra, 1998). Let the agent be in state s_t at time t. By taking an action a_t, the agent reaches a new state s_{t+1} with probability $P(s_{t+1} \mid s_t, a_t)$. An observation x_t is obtained at state s_t with probability $P(x_t \mid s_t)$. Since the state transition is dependent on the unobservable state variables in state s_t, the Markov property for observation x_t does not hold. One way to overcome the non-Markov property is to regard a history of the observation $h_t = \{(x_t, -, -), (x_{t-1}, a_{t-1}, r_{t-1}), \ldots, (x_1, a_1, r_1)\}$ as a state, and to apply an MDP formulation to such a state space. Since the capacity to maintain such a history is often limited, however, an MDP formulation whose state is a compressed representation of the history (internal state) is often used. A belief state is an example of such an internal state representation.

An optimal control problem for a POMDP can be classified into: (I) a problem in which the agent knows the environmental model, and (II) a problem in which the environmental model is unknown. In case (I), a DP approach is often used after the non-Markov property is resolved. In case (II), it is necessary to obtain the environmental model simultaneously with the resolution of the non-Markov property. This latter problem is an RL problem for a POMDP, and methods to deal with such a problem are developed by extending the conventional RL methods devised for MDPs. Case (II) can be further classified into: (IIa) the environmental model is explicitly learned, or (IIb) not. As a method for case (IIb), the temporal-difference (TD) based learning like Q-learning has often been applied to an observation state space in a POMDP. Such a method is a "naive" approach to POMDPs; it is based on a direct approximation into an MDP. In the methods in case (IIa), on the other hand, the environmental model is explicitly learned in order to calculate the current internal state of the agent.

The methods in case (IIa) can be further classified into: (IIa-i) the environmental model is dependent on the learning of the evaluation of the current state, i.e., the value function (Lin & Mitchell, 1992; McCallum, 1995), or (IIa-ii) not (Lin & Mitchell, 1992; Whitehead & Lin, 1995). In a recurrent model (Lin & Mitchell, 1992; Whitehead & Lin, 1995), which is the method in case (IIa-ii), two independent learning modules are prepared and they learn the action-value function (Q-function) and the state transition of the environment. In this model, even when the reward function changes without a change of the state transition, there is no need to train the module for the state transition learning. We also presented a similar RL scheme in which the state transition of a partial observable environment is estimated based on Bayes inference (Ishii, Yoshida, & Yoshimoto, 2002). Such an RL scheme is often called a model-based RL.

In the RL method presented in this article, action selection is done by estimating explicitly the state transition of the environment, i.e., an environmental model, while a state evaluation module approximates the value function. Since we assume the independence between the environmental model and the value function, our method belongs to (IIa-ii), that is, it is a kind of a model-based RL method. The action selection is executed based on the estimation of unobservable state variables and action prediction of the opponent agents. The action prediction is executed by a learning unit, which approximates the action selection probability of the corresponding opponent agent. These action predictors learn independently for each agent. If one agent changes its strategy, it is necessary to retrain only the corresponding unit. This is an advantage of our method in that it reduces the computational time, over that of the existing recurrent model that approximates the whole environmental model.

When our method formulated in a single-agent system is applied to a multi-agent system, it is necessary for the action predictors to adapt to the action selection probability that may change with time. Our architecture in which action predictors are individually prepared for the opponent agents can be suitable for such multi-agent systems.

3. Preparation

3.1. The card game "Hearts"

A four-player card game, Hearts, is considered a typical example of POMDP problems. Here, we explain the rules of the game used in our study.

The game Hearts is played by four players and uses the ordinary 52-card deck. There are four suits, i.e., spades (♠), hearts (♡), diamonds (♢), and clubs (♣), and there is an order of strength within each suit (i.e., A, K, Q, . . ., 2). There is no strength order among the suits. Cards are distributed to the four players so that each player has in his hand 13 cards at the beginning of the game. Thereafter, according to the rules below, each player plays a card clock-wisely in order. When each of the four players has played a card, it is called a trick. Namely, each player plays a card once in one trick. The first card played in a trick is called the leading card and the player who plays the leading card is called the leading player. A single game ends when 13 tricks are carried out.

- Except for the first trick, the winner of the current trick becomes the leading player of the subsequent trick.
- In the first trick, ♣2 is the leading card, denoting that the player holding this card is the leading player.
- Each player must play a card of the same suit as the leading card.
- If a player does not have a card of the same suit as the leading card, he can play any card. When a heart is in such a case played for the first time in a single game, the play is called "breaking hearts".
- Until the breaking hearts occurs, the leading player may not play a heart. If the leading player has only hearts, it is an exceptional case and the player may lead with a heart.
- After a trick, the player that has played the strongest card of the same suit as the leading card becomes the winner of that trick.
- Each heart equals a one-point penalty and the ♠Q equals a 13-points penalty. The winner of a trick receives all of the penalty points of the cards played in the trick.

According to the rules above,[1] a single game is played, and at the end of a single game, the score of each player is determined as the sum of the received points. The lower the score, the better.

3.2. State transition of Hearts

For the time being, we assume there are in the environment a single learning agent and three opponent agents that do not learn.

A single state transition of the game is represented by: (1) real state s that includes every card (observable and unobservable) allocation, (2) observation x for the learning agent, i.e., the cards in the agent's hand and the cards that have already been played in the past tricks and the current trick, (3) the agent's action a, i.e., a single play at his turn, and (4) strategy ϕ of each of the opponent agents. Let t indicate a playing turn of the learning agent. $t = 14$ indicates the end state of the game. At the t-th playing turn, the learning agent does not know the real state s_t, and all he can do is to estimate it by considering the history of observations and actions in the past tricks. In the following descriptions, we assume that there are three opponent agents intervening between the t-th play and the $(t + 1)$-th play of the learning agent. If the leading player of the t-th trick and that of the $(t + 1)$-th trick are different, the number of intervening opponent agents is not three. Although the following explanation can be easily extended to a general case in which the number of intervening agents is not necessarily three, this assumption is beneficial to simplifying the explanation. Between the t-th play and the $(t + 1)$-th play of the learning agent, there are three state transitions due to actions by the three opponent agents. These state transitions are indexed by t. It should be noted that this index is different from the trick index, e.g., a_t^i may be a play in the $(t + 1)$-th trick. Each of the opponent agents is also in a partial observation situation; state, observation, action and strategy at his t-th playing turn are denoted by s_t^i, x_t^i, a_t^i and ϕ_t^i, respectively, where i is the index of an opponent agent.

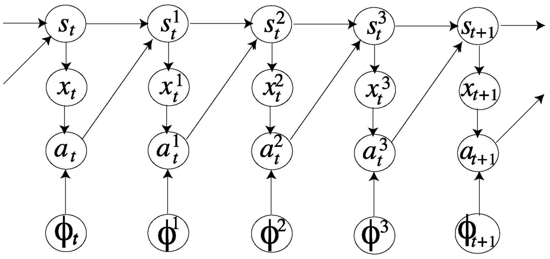

Figure 1. State transition diagram for the game Hearts. Variables s_t, x_t, a_t, and ϕ_t denote a real state, an observation, an action and a strategy, for the learning agent at his t-th playing turn, variables s_t^i, x_t^i, and a_t^i denote a real state, an observation and an action, for opponent agent M^i at the t-th turn. Variable ϕ^i does not depend on t, which corresponds to a POMDP approximation.

Let $M^i (i = 1, 2, 3)$ denote the i-th opponent agent. We assume

- assumption (a)
  Agent M^i probabilistically determines his action a_t^i for his own observation x_t^i at his t-th playing turn.

Under this assumption, the state transition between the t-th play and the $(t + 1)$-th play of the learning agent is given by

$$P(s_{t+1} \mid s_t, a_t, \Phi_t) \\ = \sum_{s_t^1, s_t^2, s_t^3} \sum_{a_t^1, a_t^2, a_t^3} \prod_{j=0}^{3} P\big(s_t^{j+1} \mid s_t^j, a_t^j\big) \prod_{i=1}^{3} \sum_{x_t^i} P\big(a_t^i \mid x_t^i, \phi_t^i\big) P\big(x_t^i \mid s_t^i\big), \quad (1)$$

where $s_t^0 = s_t$, $a_t^0 = a_t$, $s_t^4 = s_{t+1}$ and $x_t^0 = x_t$. $\Phi_t \equiv \{\phi_t^i : i = 1, 2, 3\}$, where ϕ_t^i denotes the strategy of opponent agent M^i at his t-th play.

3.3. POMDP approximation

The incomplete information game Hearts is approximated as a POMDP (see figure 1); the approximated incomplete game is called a partial observation game. In a partial observation game, it is assumed that there is only one learning agent, and the strategies of the other (opponent) agents are fixed, that is, the other agents constitute the stationary environment. Due to this POMDP approximation, $\phi^i (i = 1, 2, 3)$ does not depend on the play index t.

Since the game process of Hearts is deterministic, there are two facts:

- New state s_t^{i+1}, which is reached from a previous state s_t^i by an action a_t^i, is uniquely determined. Namely, $P(s_t^{i+1} \mid s_t^i, a_t^i)$ is 1 for a certain state and 0 for the other states.
- Observation, x_t or x_t^i, is uniquely determined at state, s_t or s_t^i. Namely, $P(x_t \mid s_t)$ or $P(x_t^i \mid s_t^i)$ is 1 for a certain observation state and 0 for the other observation states.

Since state s_t is not observable for the learning agent, it should be estimated using the history of the current game, $H_t \equiv \{(x_t, -, -), (x_{t-1}, a_{t-1}, a_{t-1}^{1,2,3}), \ldots, (x_1, a_1, a_1^{1,2,3})\}$, actions $a_t^i (i = 1, 2, 3)$ at the t-th turn, and game knowledge (game rules, etc.) K.

The transition probability for the observation state is given by

$$\begin{aligned} &P(x_{t+1} \mid a_t, \Phi, H_t, K) \\ &\quad = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} P(x_{t+1} \mid s_{t+1}) \sum_{s_t \in \mathcal{S}_t} P(s_{t+1} \mid s_t, a_t, \Phi) P(s_t \mid H_t, K), \end{aligned} \tag{2}$$

where $\mathcal{S}_t$ is the set of possible states at the t-th play of the learning agent.

From the above two facts and Eq. (1),

$$\begin{aligned} &P(x_{t+1} \mid a_t, \Phi, H_t, K) \\ &\quad = \sum_{s_t \in \mathcal{S}_t} P(s_t \mid H_t, K) \sum_{(a_t^1, a_t^2, a_t^3) \in \mathcal{A}_t^-(x_{t+1}, s_t)} \prod_{i=1}^{3} P\big(a_t^i \mid x_t^i, \phi^i, H_t, K\big). \end{aligned} \tag{3}$$

Here, $\mathcal{A}_t^-(x_{t+1}, s_t)$ denotes the set of possible (a_t^1, a_t^2, a_t^3) by which the previous state-action pair (s_t, a_t) reaches any new state whose observation state is x_{t+1} and $P(s_t \mid H_t, K)$ is a belief state. Equation (3) provides a model of the environmental dynamics.

However, the calculation in Eq. (3) has two difficulties. One is the intractability of the belief state; since the state space of the game Hearts is huge, the rigorous calculation of the summation $\sum_{s_t \in \mathcal{S}_t}$ is difficult. The other is the difficulty in retrieving the game tree; especially when there are a lot of unobservable state variables, i.e., unobservable cards, $\mathcal{A}_t^-$ is a huge set and then the calculation of the summation $\sum_{(a_t^1, a_t^2, a_t^3) \in \mathcal{A}_t^-(x_{t+1}, s_t)}$ is also difficult.

In order to cope with the former difficulty, we use the following approximation. Since the real observation x_t^i by agent M^i cannot be observed by the learning agent during the game, it is estimated using the history of the current game H_t and the game knowledge K. The estimated observation state is denoted by y_t^i. First, the probability $P(y_t^i \mid a_t, H_t, K)$ is estimated using H_t and K; the estimation method in the game Hearts will be specifically explained in Section 4.2. Using this probability, we calculate the mean estimated observation for agent M^i as

$$\hat{y}_t^i(a_t, H_t, K) \equiv \sum_{y_t^i} y_t^i P\big(y_t^i \mid a_t, H_t, K\big). \tag{4}$$

Using the mean estimated observation, the transition probability (3) is approximated as

$$\begin{aligned} &P(x_{t+1} \mid a_t, \Phi, H_t, K) \\ &\quad \approx \sum_{(a_t^1, a_t^2, a_t^3) \in \mathcal{A}_t^-(x_{t+1}, x_t)} \prod_{i=1}^{3} P\big(a_t^i \mid \hat{y}_t^i(a_t, H_t, K), \hat{\phi}^i\big). \end{aligned} \tag{5}$$

From assumption (a), each opponent agent determines its action a_t^i with probability $P(a_t^i \mid x_t^i, \phi^i, H_t, K)$. However, this action selection probability and the real observation state x_t^i are unknown for the learning agent and they should be estimated in some way. Therefore, the learning agent assumes that the action selection process is approximately done by a stochastic process that is dependent on the mean estimated observation $\hat{y}_t^i(a_t, H_t, K)$. It should be noted that the approximated strategy $\hat{\phi}^i$ in Eq. (5) is different from the real strategy ϕ^i in Eq. (3). Since the mean estimated observation $\hat{y}_t^i(a_t, H_t, K)$ incorporates the history of the current game H_t and the game knowledge K, it provides essential information of the belief state $P(s_t^i \mid a_t, H_t, K)$. Therefore, the stochastic process dependent on a discrete but unobservable observation state is approximated as a stochastic process dependent on an analog (mean) and estimated observation state. There is possibility to introduce bias in the estimation, due to the difference between the real observation state x_t^i and the estimated observation state y_t^i or its mean $\hat{y}_t^i$, and to the difference between the real action selection process $P(a_t^i \mid x_t^i, \phi^i, H_t, K)$ and the approximated action selection process $P(a_t^i \mid \hat{y}_t^i(a_t, H_t, K), \hat{\phi}^i)$. With this approximation, however, the summation $\sum_{s_t \in \mathcal{S}_t}$ is no more necessary for the calculation of the transition probability (5).

Strategy ϕ^i represents the policy that determines actions of agent M^i. The approximated policy $\hat{\phi}^i$ is represented and learned by using a function approximator. For a game finished in the past, an observation state and an action taken by an opponent agent at that state can be reproduced by replaying the game from the end to the start. In order to train the function approximator for $\hat{\phi}^i$, the input and the target output are given by $\hat{y}_t^i(a_t, H_t, K)$ and the action a_t^i actually taken by agent M^i at that turn, respectively. Since the game of Hearts is a perfect recall game and there is no probabilistic factor in the game process, x_t^i can also be reproduced and available for the input. If we use x_t^i as an input, however, the input-output relationship during the training, (x_t^i, a_t^i), and that during the playing, $(\hat{y}_t^i, a_t^i)$, have different characteristics. In order to avoid this inconsistency, we reproduce again $\hat{y}_t^i$ in the learning of the opponent agent's strategy $\hat{\phi}^i$. This learning is done according to a similar algorithm to the actor learning in the actor-critic algorithm (Barto, Sutton, & Anderson, 1983); namely, a merit function for $(\hat{y}, a)$ is updated so that an action a is selected with a higher probability for a mean estimated observation $\hat{y}$. The parameter of the function approximator represents the approximated policy $\hat{\phi}^i$ of agent M^i.

In addition, we use another approximation technique to cope with the latter difficulty, i.e., the difficulty in the calculation of the summation $\sum_{(a_t^1, a_t^2, a_t^3) \in \mathcal{A}_t^-}$. This technique will be specifically explained in Section 4.3.

3.4. Action control

According to our RL method, an action is selected based on the expected TD error, which is defined by

$$\langle \delta_t \rangle(a_t) = \langle R(x_{t+1}) \rangle(a_t) + \gamma \langle V(x_{t+1}) \rangle(a_t) - V(x_t), \tag{6}$$

where

$$\langle f(x_{t+1}) \rangle(a_t) \equiv \sum_{x_{t+1}} P(x_{t+1} \mid a_t, \Phi, H_t, K) f(x_{t+1}) \tag{7}$$

and $P(x_{t+1} \mid a_t, \Phi, H_t, K)$ is given by Eq. (5). The expected TD error considers the estimation of the unobservable states and the strategies of the other agents.

Using the expected TD error, the action selection probability is determined as

$$P(a_t \mid x_t) = \frac{\exp(\langle \delta_t \rangle (a_t)/T_m)}{\sum_{a_t \in \mathcal{A}} \exp(\langle \delta_t \rangle (a_t)/T_m)}, \tag{8}$$

where T_m is a parameter controlling the action randomness.

Our RL method uses the TD error expected with respect to the estimated transition probability for the observation state. An action is then determined based on the estimated environmental model. Such an RL method is often called a model-based RL method. Our idea that the action priority is determined based on the expected TD error is similar to that in the prioritized sweep algorithm by Moore and Atkeson (1993), which has been reported to be effective in problems consisting of a large number of states.

3.5. Actor-critic algorithm

Although the actor-critic algorithm (Barto, Sutton, & Anderson, 1983) is *not* used in our RL method, it is briefly introduced here for the convenience of explanation. According to the actor-critic algorithm, the critic maintains the value function $V(x_t)$ that evaluates state x_t at the t-th turn of the learning agent, and the actor determines its action a_t based on a merit function $U(x_t, a_t)$.

The critic calculates the TD error for a given state transition for observable states:

$$\delta_t = R(x_{t+1}) + \gamma V(x_{t+1}) - V(x_t), \tag{9}$$

where $R(x_{t+1})$ is the reward function that is assumed to be dependent only on the observation state x_{t+1}. In the case of Hearts, the reward function represents the (negative) penalty points that the learning agent receives at the t-th trick.

Using the TD error, the critic updates the value function and the actor updates the merit function as

$$V(x_t) \leftarrow V(x_t) + \eta_c \delta_t \tag{10a}$$

$$U(x_t, a_t) \leftarrow U(x_t, a_t) + \eta_a \delta_t, \tag{10b}$$

where η_c and η_a are the learning rates for the critic and the actor, respectively.

Using the merit function, the actor selects an action according to the Boltzmann policy

$$P(a_t \mid x_t) = \frac{\exp(U(x_t, a_t)/T_e)}{\sum_{a_t \in \mathcal{A}} \exp(U(x_t, a)/T_e)}, \tag{11}$$

where T_e is a parameter controlling the action randomness and $\mathcal{A}$ denotes the set of possible actions.

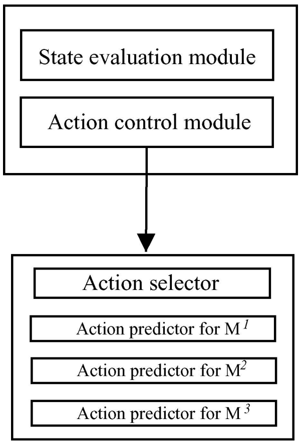

Figure 2. The architecture that realizes our RL method. It consists of a state evaluation module and an action control module. The action control module consists of three action predictors and an action selector.

4. Method

This section describes in detail our RL method. The architecture implementing our method roughly consists of two modules (see figure 2): a state evaluation module and an action control module. The action control module consists of three action predictors each corresponding to each of the three opponent agents and one action selector.

4.1. State evaluation module

The state evaluation module has the same role as the critic in the actor-critic algorithm. In our previous preliminary study, the input and the output of the state evaluation module were the current observation state x_t and the corresponding value function $V(x_t)$, respectively (Matsuno et al., 2001). With this implementation, however, the input dimension was equal to or larger than the number of cards, and the approximation of the value function was time consuming even with a function approximator. Therefore, we use a feature extraction technique. An input to the function approximator, p_t, is given mainly by the transformation from an observation state x_t as follows.

- $p_t(1)$: the number of club cards that have been played in the current game, or are held by the learning agent.
- $p_t(2)$: the number of diamond cards that have been played in the current game, or are held by the learning agent.
- $p_t(3)$: the number of spade cards (♠2, . . . , ♠J) that have been played in the current game, or are held by the learning agent.
- $p_t(4)$, $p_t(5)$ and $p_t(6)$: the probability that agent M^1, M^2 and M^3 have the ♠Q, respectively.

- $p_t(7)$: the status of the ♠K.
- $p_t(8)$: the status of the ♠A.
- $p_t(9)$ to $p_t(21)$: the status of each of the heart cards.
- $p_t(22)$ to $p_t(25)$: a bit sequence representing who is the leading player in the current trick.

Since the most important card is the ♠Q in the game of Hearts, we use three dimensions to represent its predictive allocation. The game rules tell us the following facts.

1. If agent M^i did not play a spade card when the leading card was a spade card in a past trick of the current game, $p_t(i+3)$ is zero.
2. $p_t(4) + p_t(5) + p_t(6) = 1$.

Under the limitation from these two facts, the probability that agent M^i has the ♠Q, $p_t(i+3)$, is calculated as a uniform probability. The status of the ♠K, the ♠A, or a heart card is represented by one of three values, -1, 0 or 1, corresponding to the cases when the card has already been played in the current game, when it is held by the opponent agents, or when it is held by the learning agent, respectively. The bit sequence represents the playing order in the current trick. For example, when the learning agent is the second player in the current trick (the t-th playing turn of the learning agent), $[p_t(22), p_t(23), p_t(24), p_t(25)] = [0, 1, 0, 0]$.

In this study, the state evaluation module is trained so as to approximate $V(p_t)$ for an input p_t. This learning is done by Eqs. (9) and (10a) where x_t and x_{t+1} are replaced by p_t and p_{t+1}, respectively. It should be noted that the value function represented by the state evaluation module depends not only on the observation state x_t but partly on the estimation of the unobservable state; namely, $p_t(4)$, $p_t(5)$ and $p_t(6)$ reflect the estimation.

4.2. Action predictor

In the action selection module, there are three action predictors. The action predictor for agent M^i predicts a card played by that agent, in a similar manner to the action selection by the actor in the actor-critic algorithm. In order to predict an action by agent M^i at his t-th turn, the i-th action predictor calculates a merit function value $U^i(\hat{y}^i_t(a_t, H_t, K), a^i_t)$ for the mean estimated observation $\hat{y}^i_t(a_t, H_t, K)$ and a possible action a^i_t. After calculating the merit value for every possible action, an action a^i_t is selected with the predicted probability

$$P\big(a^i_t \mid \hat{y}^i_t(a_t, H_t, K), \phi^i\big) = \frac{\exp\big(U^i\big(\hat{y}^i_t(a_t, H_t, K), a^i_t\big)/T^i\big)}{\sum_{a^i_t \in \mathcal{A}^i} \exp\big(U^i\big(\hat{y}^i_t(a_t, H_t, K), a^i_t\big)/T^i\big)}. \tag{12}$$

Here, $\mathcal{A}^i$ denotes the set of possible actions for agent M^i, and T^i is a constant parameter that denotes the assumed randomness of the action selection of agent M^i.

When training the action predictor for agent M^i, the merit function, $U^i(\hat{y}^i_t(a_t, H_t, K), a^i_t)$ is updated similarly to the actor learning (Eqs. (9) and (10b)). $\hat{y}^i_t(a_t, H_t, K)$ is reproduced by replaying a past game, and a_t^i is the action actually taken by agent M^i at his t-th play in the past game.

We use a function approximator for representing the merit function. In order to faithfully implement the above learning of the action predictor, however, the dimensions of the input and output of the function approximator become equal to or larger than the number of cards. This learning is difficult and often needs a large amount of computation time even with an efficient function approximator. Therefore, we use a feature extraction technique as well as in the state evaluation module.

An input to the function approximator, q_t^i, is given by the transformation from the mean estimated observation $\hat{y}_t^i$ as follows.

- $q_t^i(1)$: if the leading card is a club card, the expected number of club cards held by agent M^i, which are weaker than the strongest card already played in the current trick, otherwise zero.
- $q_t^i(2)$: if the leading card is a club card, the expected number of club cards held by agent M^i, which are stronger than the strongest card already played in the current trick, otherwise the expected number of club cards held by the agent.
- $q_t^i(3)$: similar to $q_t^i(1)$, but the suit is diamond.
- $q_t^i(4)$: similar to $q_t^i(2)$, but the suit is diamond.
- $q_t^i(5)$: if the leading card is a spade card, the expected number of spade cards (♠2, ..., ♠J) held by agent M^i, which are weaker than the strongest card already played in the current trick, otherwise zero.
- $q_t^i(6)$: if the leading card is a spade card, the expected number of spade cards (♠2, ..., ♠J) held by agent M^i, which are stronger than the strongest card already played in the current trick, otherwise the expected number of spade cards (♠2, ..., ♠J) held by the agent.
- $q_t^i(7)$: the expectation value for that agent M^i has the ♠Q.
- $q_t^i(8)$: the expectation value for that agent M^i has the ♠K.
- $q_t^i(9)$: the expectation value for that agent M^i has the ♠A.
- $q_t^i(10)$ to $q_t^i(22)$: the expectation value for that agent M^i has each of the heart cards.
- $q_t^i(23)$ to $q_t^i(26)$: a bit sequence representing who is the leading player in the current trick.

Let $C_t^i(♠\mathrm{Q})$ be 1 or 0 when agent M^i has or does not have, respectively, the ♠Q in his hand just before his t-th turn, for example. The expectation value of the binomial variable $C_t^i(♠\mathrm{Q})$ is equivalent to the probability that agent M^i has the ♠Q in his hand:

$$\hat{C}_t^i(♠\mathrm{Q} \mid a_t, H_t, K) = P\big(C_t^i(♠\mathrm{Q}) = 1 \mid a_t, H_t, K\big). \tag{13}$$

The game rules tell us the following facts.

1. If agent M^i did not play a card of the same suit as the leading card in a past trick of the current game, M^i does not have at present any card of this suit.
2. The cards, except for those held by the learning agent and those that have already been played in the current game, may exist in the hand of agent M^i.

Under the limitation from these two facts, the card existence probability in the hand of agent M^i is assumed to be uniform. The value $\hat{C}^i_t(\text{a-card} \mid a_t, H_t, K) \in [0, 1]$, which represents the expectation value for that agent M^i has 'a-card' in his hand, is then calculated with respect to the distribution. The values $q^i_t(7), \ldots, q^i_t(22)$ correspond to $\hat{C}^i_t(\spadesuit\text{Q} \mid a_t, H_t, K), \ldots, \hat{C}^i_t(\heartsuit\text{A} \mid a_t, H_t, K)$, respectively. The values $q^i_t(1), \ldots, q^i_t(6)$ are calculated by using $\hat{C}^i_t(\clubsuit 2 \mid a_t, H_t, K), \ldots, \hat{C}^i_t(\spadesuit\text{J} \mid a_t, H_t, K)$. Namely, q^i_t is given by the transformation from the estimated card existence probability $\hat{C}^i_t$. It should be noted that $\hat{C}^i_t$ is similar to the mean expected observation $\hat{y}^i_t$. An input to the function approximator is thus given by the transformation from the mean estimated observation $\hat{y}^i_t$.

The action predictor is trained so as to output the following 26-dimensional vector:

1. $r^i_t(1)$: if the leading card is a club card, the merit value for that agent M^i plays a weaker club card than the strongest card already played in the current trick.
2. $r^i_t(2)$: if the leading card is a club card, the merit value for that agent M^i plays a club card that is stronger than the strongest card already played in the current trick, and the weakest in the hand of M^i.
3. $r^i_t(3)$: if the leading card is a club card, the merit value for that agent M^i plays a club card that is stronger than the strongest card already played in the current trick, and neither the weakest nor the strongest in the hand of M^i.
4. $r^i_t(4)$: if the leading card is a club card, the merit value for that agent M^i plays a club card that is stronger than the strongest card already played in the current trick, and the strongest in the hand of M^i.
5. $r^i_t(5)$: similar to $r^i_t(1)$, but the suit is diamond.
6. $r^i_t(6)$: similar to $r^i_t(2)$, but the suit is diamond.
7. $r^i_t(7)$: similar to $r^i_t(3)$, but the suit is diamond.
8. $r^i_t(8)$: similar to $r^i_t(4)$, but the suit is diamond.
9. $r^i_t(9)$: if the leading card is a spade card, the merit value for that agent M^i plays a weaker spade card among $\spadesuit 2, \ldots, \spadesuit\text{J}$ than the strongest card already played in the current trick.
10. $r^i_t(10)$: if the leading card is a spade card, the merit value for that agent M^i plays a stronger spade card among $\spadesuit 2, \ldots, \spadesuit\text{J}$ than the strongest card already played in the current trick.
11. $r^i_t(11)$: the merit value for that agent M^i plays the $\spadesuit$Q.
12. $r^i_t(12)$: the merit value for that agent M^i plays the $\spadesuit$K.
13. $r^i_t(13)$: the merit value for that agent M^i plays the $\spadesuit$A.
14. $r^i_t(14)$ to $r^i_t(26)$: the merit value for that agent M^i plays each of the heart cards.

The input and output of the function approximator for the i-th action predictor are q^i_t and r^i_t, respectively. From the 26-dimensional output r^i_t, the merit value of every possible card, i.e., $U^i(q^i_t, a^i_t)$ for every possible action a^i_t, is calculated. The 26-dimensional output r^i_t focuses on which player becomes the winner of the t-th trick. Since the specific cards that will be played in that trick is necessary for evaluating $V(p_{t+1})$ by the state evaluation module, however, we transform r^i_t into $U^i(q^i_t, a^i_t)$ and evaluate every possible combination of cards that will be played in the i-th trick. If there are more than one possible cards to be played in this transformation, the merit values for those cards are set at the same value, e.g., $U^i(q_t^i, ♣8) = U^i(q_t^i, ♣9) = r_t^i(3)$ might be such a case. As a consequence, both of the input dimension and the output dimension of the function approximator are 26. This dimension number is much smaller than that in our previous study (Matsuno et al., 2001). It is expected that this dimension reduction accelerates the learning of the action predictor and hence accelerates the strategy acquisition of the learning agent.

Here, the prediction by the action predictor is summarized. The action predictor for agent M^i calculates the estimated card existence probability $\hat{C}_t^i$, and then the input to the function approximator, q_t^i, is calculated from $\hat{C}_t^i$. In the actual implementation, we directly calculate q_t^i without calculating $\hat{C}_t^i$. This calculation corresponds to the process expressed by Eq. (4). Then, the function approximator of the action predictor outputs the reduced merit function r_t^i for the input q_t^i. After that, r_t^i is transformed into the merit function $U^i(q_t^i, a_t^i)$, and then a possible action is selected by Eq. (12), in which $U^i(\hat{y}_t^i, a_t^i)$ is replaced by $U^i(q_t^i, a_t^i)$.

4.3. Action selector

The action selector determines an action based on the Boltzmann selection rule (8). In order to obtain the expected TD error (6), it is necessary to estimate the transition probability $P(x_{t+1} \mid a_t, \Phi, H_t, K)$, as specified in Eq. (5). In order to calculate Eq. (5), it is necessary to estimate $\hat{y}_t^i(a_t, H_t, K)$, as shown in Eq. (4) and then to calculate $P(a_t^i \mid \hat{y}_t^i(a_t, H_t, K), \phi^i)$. The estimation of $\hat{y}_t^i(a_t, H_t, K)$ is replaced by the estimation of $q_t^i(a_t, H_t, K)$, and the calculation of $P(a_t^i \mid \hat{y}_t^i(a_t, H_t, K), \phi^i)$ is approximately done by Eq. (12). By producing every possible combination of actions, (a_t^1, a_t^2, a_t^3), Eq. (5) is calculated, and then the expected TD error is obtained using the probability (5) for every possible new observation state x_{t+1}.

Especially when there are a lot of cards that can be played in the t-th trick, however, the complete retrieval for every possible combination of cards played in the trick and for every possible new observation state is difficult. This difficulty partly corresponds to the difficulty of the calculation of the summation $\sum_{(a_t^1, a_t^2, a_t^3) \in \mathcal{A}_t^-}$ in Eq. (3). In order to overcome this difficulty, we use the following pruning technique. For each possible action for agent M^i at his t-th play, a_t^i, the action predictor calculates a merit value $U^i(q_t^i, a_t^i)$ for a pair of the reduced mean estimated observation $q_t^i(a_t, H_t, K)$ and action a_t^i. After that, by calculating the mean and the standard deviation (s.d.) of the merit values over the possible actions, a probability for selecting an action whose merit value is smaller than (mean) − (s.d.) is determined as 0. Namely, a state transition due to an action whose merit value is fairly small is dropped in the further evaluation; this introduces pruning within the game tree, in order to obtain efficiently the summation in Eqs. (5) and (7).

For the remaining actions, the action probability is determined as

$$P\left(a_t^i \,\middle|\, q_t^i(a_t, H_t, K), \phi^i\right) \approx \frac{\exp\left(U^i\left(q_t^i(a_t, H_t, K), a_t^i\right)/T^i\right)}{\sum_{a_t^i \in \mathcal{A}^{i-}} \exp\left(U^i\left(q_t^i(a_t, H_t, K), a_t^i\right)/T^i\right)} \tag{14}$$

instead of Eq. (12), where $\mathcal{A}^{i-}$ denotes the set of actions that are not dropped.

4.4. Function approximator

If the learning uses function approximators, the merit functions and the value function for an unknown state can be estimated owing to the generalization ability of the function approximators. Since the state space of a realistic problem like that of the game Hearts is huge and it is difficult for the learning system to experience every possible state, the generalization ability of function approximators is very important.

In this study, we use normalized Gaussian networks (NGnet) (Moody & Darken, 1989) as function approximators. NGnet is defined as

$$O = \sum_{k=1}^{m} \frac{G_k(I)}{\sum_{l=1}^{m} G_l(I)} (W_k I + b_k) \tag{15a}$$

$$G_k(I) = (2\pi)^{-N/2} \mid \Sigma_k|^{-1/2} \exp\left[-\frac{1}{2}(I - \mu_k)' \Sigma_k^{-1} (I - \mu_k) \right], \tag{15b}$$

where I denotes an N-dimensional input vector and O denotes an N_a-dimensional output vector, m denotes the number of units, Σ_k is an $N \times N$ covariance matrix, μ_k is an N-dimensional center vector, W_k is an $N_a \times N$ weight matrix, and b_k is an N_a-dimensional bias vector, for the k-th unit. Prime ($'$) denotes a transpose.

The NGnet can be defined as a probabilistic model, and its maximum likelihood inference is done by an on-line expectation-maximization (EM) algorithm (Sato & Ishii, 2000). The on-line EM algorithm is based on a stochastic gradient method, and is faster than gradient methods. Therefore, the learning of the action predictors is so fast that our RL method can be applicable to a situation where the strategies of the opponent agents change with time. Although the approximation accuracy is dependent on the number of units, m in Eq. (15), its automatic determination method based on the probabilistic interpretation is implemented in the on-line EM algorithm (Sato & Ishii, 2000).

5. Computer simulations

During a single game, each action of the learning agent is determined by the action control module that includes the three action predictors. Concurrently with this action control, the state evaluation module is trained according to the TD-learning (Eqs. (9) and (10a)) for the transformed observation p_t. After a single game ends, the three action predictors are trained by using a reproduced mean estimated observation $\hat{y}_t^i$ and the action actually taken at that time, by replaying the previous single game. This procedure is called a single training game, and the learning proceeds by repeating training games. Since we use an efficient on-line algorithm for training the function approximators, it is expected that our RL method adapts gradually to the strategies of the opponent agents, not only when they are stationary but also when they change within a slower time-scale than the adaptation by the on-line learning.

5.1. Single agent learning in a stationary environment

We carried out computer simulation experiments using one learning agent based on our RL method and three rule-based opponent agents.

The rule-based agent has more than 50 rules so that it is an “experienced” level player of the game Hearts. The penalty ratio was 0.41 when an agent who only took out permitted cards at random from its hand challenged the three rule-based agents. The penalty ratio is the ratio of penalty points acquired by the learning agent to the total penalty points of the four agents. That is, a random agent acquired about 2.1-fold penalty points of rule-based agents on average.

Figure 3 shows the learning curve of an agent trained by our RL method when it challenged the three rule-based agents. This learning curve is an average over twenty learning runs, each of which consisted of 120,000 training games. After about 80,000 games playing with the three rule-based agents, our RL agent came to acquire a smaller penalty ratio than the rule-based agents. Namely, the RL agent got stronger than the rule-based agents, which is statistically significant as the top panel in figure 3 shows. By observing the results of the twenty learning runs (detailed data not shown), we have found that the automatic strategy acquisition can be achieved in a stable fashion by our RL method.

In our previous study, an agent trained by our model-based RL method could not beat the rule-based agents after 5,000 learning games (Matsuno et al., 2001). The present RL method is similar to our previous preliminary model-based RL method in principle, but includes newly devised feature extraction techniques used in the state evaluation module and the three action predictors. Due to the dimension reduction by the feature extraction techniques, the learning process has been accelerated much and then 120,000 training games could be executed to train the learning agent.

5.2. Learning of multiple agents in a multi-agent environment

So far, our RL method has been based on the POMDP approximation, namely, it is assumed that there is only one learning agent in the environment. In this section, we try to apply our RL method directly to multi-agent environments, in which there are multiple learning and hence dynamic agents.

Figure 4 shows the result when one learning agent trained by our RL method, one learning agent based on the actor-critic algorithm, and two rule-based agents played against each other. In order to clarify the advantage of our model-based RL method, regardless of the feature extraction techniques we use, this actor-critic agent also incorporates feature extraction techniques for its actor and critic, which are similar to those used in our RL method. Due to the feature extraction techniques, this new actor-critic agent learns much faster than an actor-critic agent without the feature extraction (Matsuno et al., 2001; data not shown). Although the average penalty ratio of our RL agent became smaller than those of the rule-based agents after about 50,000 training games, the learning agent trained by the actor-critic algorithm was not improved much. This result implies that our model-based approach within the POMDP formulation is more efficient than a model-free approach, i.e., the actor-critic algorithm.

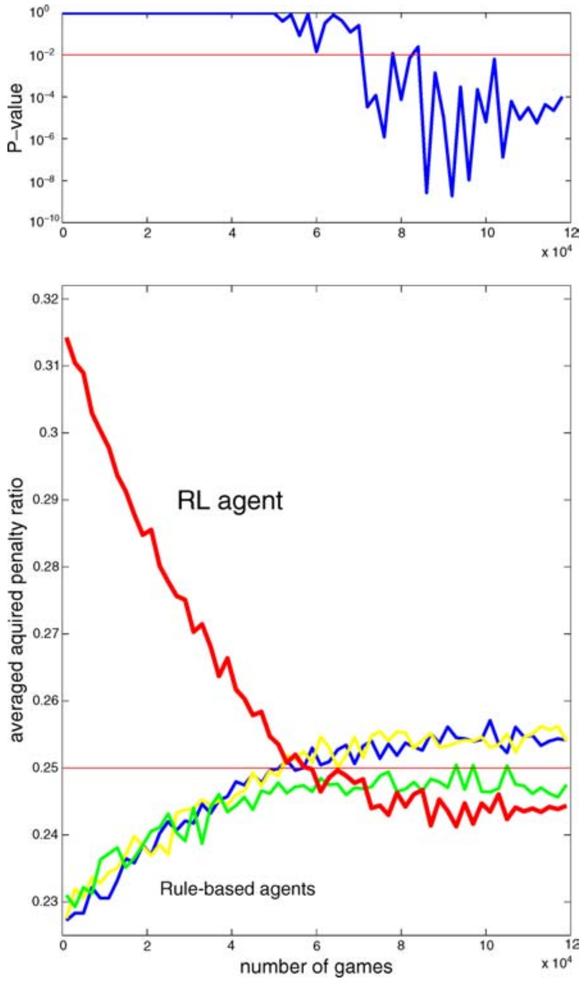

Figure 3. A computer simulation result using one learning agent trained by our RL method and three rule-based agents. *Bottom panel*: Abscissa denotes the number of training games, and ordinate denotes the penalty ratio acquired by each agent, which is smoothed by using 2,000 games just before that number of training games. We executed twenty learning runs, each consisting of 120,000 training games, and each line in the figure represents the average over the twenty runs. If the four agents have equal strength, the penalty ratio becomes 1/4, which is denoted by the horizontal line in the figure. *Top panel*: *P*-values of the statistical t test. The null hypothesis is "the RL agent is equal in strength to the rule-based agents", and the alternative hypothesis is "the RL agent is stronger than the rule-based agents". The statistical test was done independently at each point on the abscissa. The horizontal line denotes the significance level of 1%. Because we have twenty samples, the t test was applied here. The non-parametric Wilcoxon's rank-sum test also showed a similar result (not shown). After about 70,000 training games, the RL agent significantly ($p < 0.01$) became stronger than the rule-based agents.

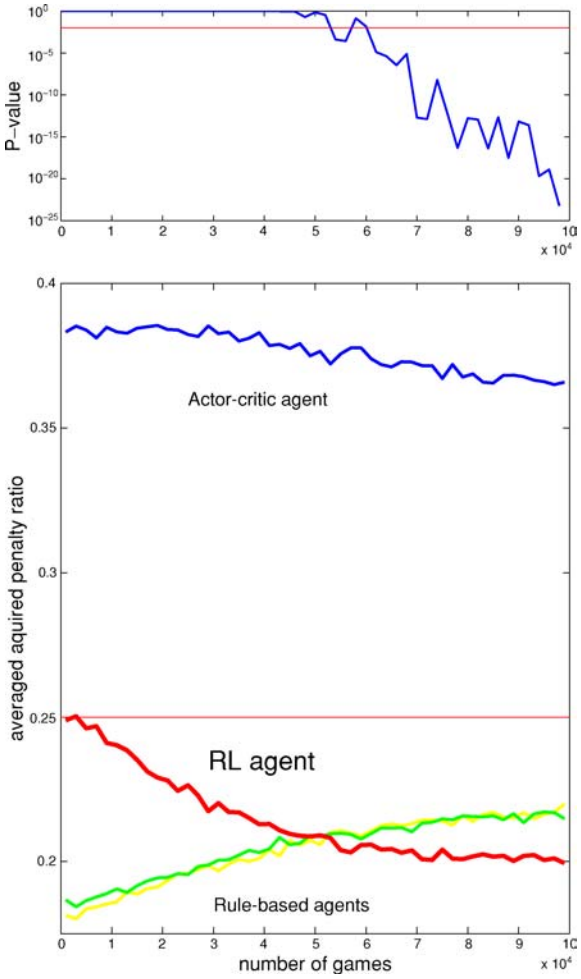

Figure 4. A computer simulation result when one learning agent trained by our RL method, one learning agent based on the actor-critic algorithm, and two rule-based agents played against each other. *Bottom panel*: Abscissa denotes the number of training games, and ordinate denotes the penalty ratio acquired by each agent, which is smoothed by using 2,000 games just before that number of training games. *Top panel*: P-values of the statistical t test. The null and alternative hypotheses are the same as those in figure 3. After about 60,000 training games, the RL agent significantly ($p < 0.01$) became stronger than the rule-based agents. The actor-critic agent was significantly ($p < 0.01$) weaker than the rule-based agents throughout the training games (figure now shown).

Figure 5 shows the result when two learning agents trained by our RL method and two rule-based agents played with each other. In this simulation, the sitting positions of the four agents were fixed throughout the training run. After about 50,000 training games, both of the two learning agents became stronger than the rule-based agents; this is statistically significant as the top panel in figure 5 shows.

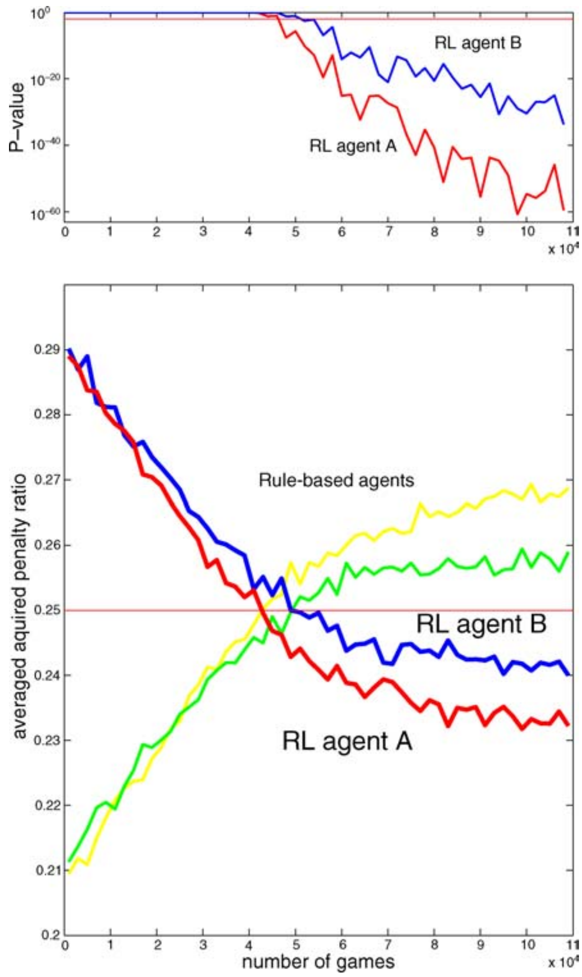

Figure 5. A computer simulation result when two learning agents trained by our RL method and two rule-based agents played against each other. The meanings of the axes are the same as those in figure 4. After about 50,000 training games, the two RL agents significantly ($p < 0.01$) became stronger than the two rule-based agents. In this simulation, the sitting positions of the four agents were fixed throughout the training run. This is the reason why the RL agent A got stronger than the RL agent B.

These two simulation results, figures 4 and 5, show that our RL method can be applied to the concurrent learning of multiple agents in a multi-agent environment. This applicability is partly attributed to the fast learning by efficient function approximators. In our RL method, we individually prepare an action predictor that approximates the policy of each opponent agent. We consider this implementation is suitable for application to multi-agent environments. Each action evaluator is able to deal with the characteristics of the corresponding opponent agent independently of the other opponent agent. In addition, if the strategy of one agent changes, it is enough for a single function approximator to adapt to the change independently. It is then expected that the RL process is stable even in a concurrent learning setting in a multi-agent environment.

Although the learning agents trained by our RL method got stronger than the rule-based agents, one may think that the RL agents adapted themselves such to pick fault of the rule-based agents. To examine the general strength of the learning agents, they were evaluated by playing against a human expert player (the designer of the rule-based agent). Figure 6 shows the result; this figure shows that the learning agents successfully acquired general strategy to become as strong as the human expert player.

6. Discussion

The automatic player for the card game "Bridge", called "GIB" (Ginsberg, 2001), resolves the partial observability using a sampling technique. In the GIB, the distribution of the unobservable cards is assumed to be random, and a possible allocation is sampled from the distribution. Using a large number of such samples and their evaluation, the expected evaluation over the samples are calculated, and then the optimal action is selected so as to maximize the expected evaluation. Therefore, a lot of samples are necessary for the determination of a single action.

In our RL method, on the other hand, the strategies of opponent agents are obtained by function approximators, which are trained by using a reproduced mean expected observation state and the action actually taken in the past. Therefore, the learning of the environmental dynamics is done by experiencing a lot of games. That is, the sampling used for the model estimation is equivalent to actual game playing. In the proposed method, it is necessary for the expected TD error to be able to calculate the expectations of the reward and the value with respect to the next observation state, as can be seen in Eq. (7). One of the advantages of our method is that sampling is not necessary for these expectations, i.e., the resolution of the partial observability; instead, we use function approximators to calculate them. The benefit derived is a reduction of the computational time.

However, we used several important approximations, one of which is that the policy of the opponent agents can be described by the mean expected observation state (Eq. (5)), and the mean observation state is also estimated from the observation of the learning agent (Eq. (4)). This approximation may introduce a bias (inaccuracy) to the estimation of the expected TD error. Since we deal with a realistic POMDP-RL problem comprised of a huge number of possible states, however, a reduction of the computation time is crucial. The computer simulation results showed that our RL method is applicable to such a realistic problem and also to a more difficult problem within a multi-agent system.

Although RL methods have been successfully applied to perfect information games, e.g., to the game Backgammon (Tesauro, 1994), there have been few applications to imperfect information games. One reason is the state transition in an imperfect information game does not have a Markov property, while the conventional RL methods devised for MDPs is not suitable for such non-Markov problems.

This article aimed at presenting an RL method applicable to realistic multi-agent problems, and we have successfully created an experienced-level player of the game Hearts.

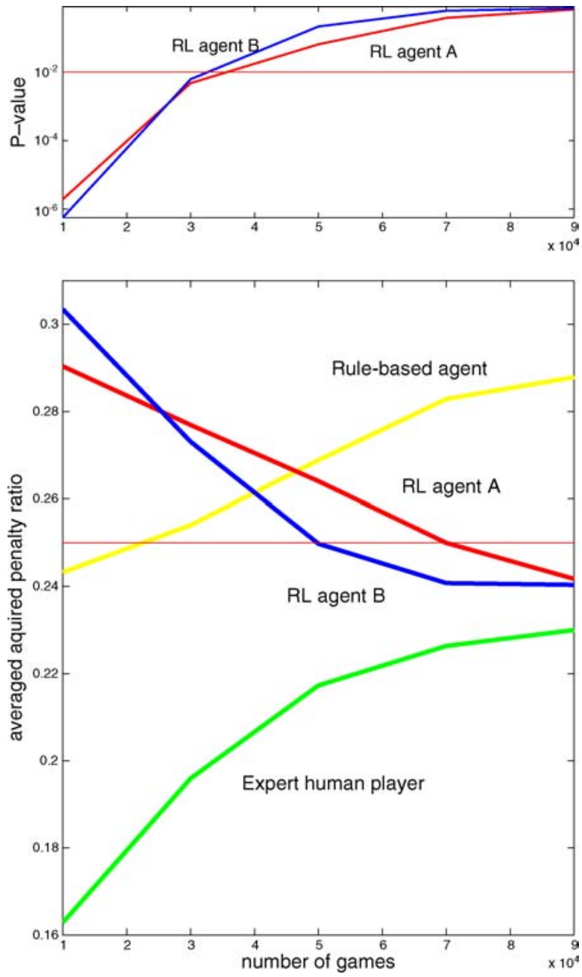

Figure 6. In the same training condition as that in figure 5, the two RL agents (the RL agents A and B in Figure 5) were evaluated by playing 100 test games against a rule-based agent and a human expert player. After 10,000, 30,000, 50,000, 70,000 and 90,000 training games, 100 test games were done. We repeated the training and evaluation run above twice. *Bottom panel*: the meanings of the axes are the same as those in figure 4. Each point denotes the average of 200 (2 × 100) test games. *Top panel*: P-values of the statistical t test. The null hypothese is "the human expert is equal in strength to the RL agent A or B", and the alternative hypothesis is "the human expert is stronger than the RL agent A or B". The horizontal line denotes the significance level of 1%. After 50,000 training games, the human expert was not significantly stronger than the RL agent A or B, with the significance level of 1%.

There have been a lot of multi-agent RL studies applied to simplified problems (Littman, 1994; Hu & Wellman, 1998; Nagayuki, Ishii, & Doya, 2000; Salustowicz, Wiering, & Schmidhuber, 1998; Sandholm & Crites, 1995; Sen, Sekaran, & Hale, 1994; Tan, 1993). One of the existing realistic multi-agent RL studies is an application to an elevator dispatch problem (Crites, 1996; Crites & Barto, 1996), while it was suggested that the performance was not good when there was unobservable information.

In this study, we have presented a model-based RL method in order to deal with large-scale POMDPs. When we assume that there is only one learning agent in a multi-agent environment, an optimal control problem in such an environment is formulated as a POMDP. In order to overcome the information incompleteness that inevitably occurs in a multi-agent problem, we used an estimation of the unobservable state variables and the policy prediction of the other agents. The experimental results showed that our RL method can be applied to a realistic multi-agent problem, in which there are more than one learning agent in the environment.

One of the features of our RL method is that we prepare an individual action predictor for each of the other agents. If the strategy of the other agents are similar to each other, one action predictor will be enough and its learning will be much faster than our method. Although our RL method assumes a single-agent POMDP in principle, however, our motivation is in the learning scheme in multi-agent environments. It is considered that the learning of each agent's characteristics, e.g., idiosyncrasies, is important in a multi-agent environment.

Our RL method is significantly dependent on the opponent agents. In our simulation experiments, we prepared rule-based agents that were fairly strong. Whether or not our RL method is also effective in a self-play problem, in which there are only learning agents that are initially very weak, is an important future issue.

7. Conclusion

This article presented an RL method applicable to an n-players ($n \geq 2$) non-cooperative, finite-state, incomplete-information game "Hearts". The presented method is based on the formulation of a POMDP, and the information incompleteness is resolved based on the distribution estimation of the unobservable cards and the strategy prediction of the other agents. Although the rigorous solution of a POMDP and the learning of the environmental model need heavy computation, the approximations introduced in the proposed method were shown to successfully reduce the computation time so that the RL can be executed as a computer simulation. As a consequence, a learning agent trained by our method became an experienced-level player of the game Hearts. The proposed RL method is a single-agent learning that assumes the strategies of opponent agents are fixed. However, experimental results showed that the method is potential to deal with a multi-agent system in which there were two learning agents. As a future work, we will extend our RL method so as to make it applicable to other multi-agent coordination/competition problems.

Acknowledgment

The authors wish to thank the editor and the reviewers for their valuable comments in improving the quality of this paper. This study was partly supported by Grant-in-Aid for Scientific Research (B) (No. 16014214) from Japan Society for the Promotion of Science.

Note

1. A standard game setting of Hearts has some other rules. For example, each player selects two or three cards from his hand to pass to another player before the first trick. If such rules are added, the learning agent is required to acquire complicated strategies in order to cope with them. In this study, we simplify the game setting and make the learning easier. However, still the learning is not easy, because the state space of the game of Hearts is huge.

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst., Man. & Cybern., 13*, 834–846.

Blair, J. R. S., Mutchler, D., & Lent, M. (1995). Perfect recall and pruning in games with imperfect information. *Computational Intelligence, 12*, 131–154.

Crites, R. H. (1996). Large-scale dynamic optimization using teams of reinforcement learning agents. Ph.D. thesis, University of Massachusetts, Amherst.

Crites, R. H., & Barto, A. G. (1996). Elevator group control using multiple reinforcement learning agents. *Machine Learning, 33*, 235–262.

Ginsberg, M. (2001). Gib: Imperfect information in a computationally challenging fame. *Journal of Artificial Intelligence Research, 14*, 303–358.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 242–250).

Ishii, S., Yoshida, W., & Yoshimoto, J. (2002). Control of exploitation-exploration meta-parameter in reinforcement learning. *Neural Networks, 15*, 665–687.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Lin, L.-J., & Mitchell, T. (1992). Memory approaches to reinforcement learning in non-markovian domains. Tech. rep., CMU-CS-92-138.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning* (pp. 157–163).

Matsuno, Y., Yamazaki, T., Matsuda, J., & Ishii, S. (2001). A multi-agent reinforcement learning method for a partially-observable competitive game. In *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 39–40).

McCallum, A. (1995). Reinforcement learning with selective perception and hidden state. Ph.D. thesis, Univercity of Rochester.

Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation, 1*, 281–294.

Moore, A., & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning, 13*, 103–130.

Nagayuki, Y., Ishii, S., & Doya, K. (2000). Multi-agent reinforcement learning: An approach based on the other agent's internal model. In *Proceedings of the Fourth International Conference on MultiAgent Systems* (pp. 215–221).

Salustowicz, R. P., Wiering, M. A., & Schmidhuber, J. (1998). Learning team strategies: Soccer case studies. *Machine Learning, 33*, 263–282.

Sandholm, T. W., & Crites, R. H. (1995). Multiagent reinforcement learning in the iterated prisoner's dilemma,. *Biosystems, 37*, 147–166.

Sato, M., & Ishii, S. (2000). On-line em algorithm for the normalized gaussian network. *Neural Computation, 12*, 407–432.

Sen, S., Sekaran, M., & Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 426–431).

Sutton, R., & Barto, A. (Eds.). (1998). *Reinforcement learning: An introduction*. MIT Press.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 330–337).

Tesauro, G. J. (1994). Td-gammon, a self-teaching backgammon program, achieves masterlevel play. *Neural Computation, 6*, 215–219.

Whitehead, S., & Lin, L.-J. (1995). Reinforcement learning of non-markov decision processes. *Artificial Intelligence, 73*, 271–306.

Received March 29, 2002
Revised September 15, 2004
Accepted October 27, 2004