# HIERARCHICAL MULTIAGENT REINFORCEMENT LEARNING IN MARKOV GAMES

*Ville Könönen*

Neural Networks Research Centre
Helsinki University of Technology
P.O.Box 5400, FI-02015 TKK, FINLAND
ville.kononen@tkk.fi

## ABSTRACT

Interactions between intelligent agents in multiagent systems can be modeled and analyzed by using game theory. The agents select actions that maximize their utility function so that they also take into account the behavior of the other agents in the system. Each agent should therefore utilize some model of the other agents. In this paper, the focus is on the situation which has a temporal structure and in which the exact form of the interaction between the learning agents is initially unknown and should be learned from the experience.

## 1. INTRODUCTION

Game theory provides a framework for constructing and analyzing various social interactions between intelligent and rational decision makers. In many cases, the roles of the decision makers are not identical; some decision makers may know their opponents' action selections and some may not. By utilizing this information, space and computational requirements of the methods utilizing game theory can be considerably reduced.

In this paper, the focus is on the situation in which the exact structure of the task is initially unknown and should be learned from the experience. For making optimal decisions in multiagent systems, actions of other agents should be taken into account and therefore it is essential to be able to model the behavior of the opponents. Particularly, we concentrate on problems with a temporal structure. Methods based on Markov Decision Processes (MDPs) provide a solution to such sequential decision problems and our proposed methods lean on the multiagent extensions of MDPs, i.e. Markov Games (MGs). We propose a temporal difference method for complex learning problems of three or more learning agents.

The first learning method for multistate Markov games was proposed by Littman in [1]. He introduced an off-policy method for Markov games with two players and a zero-sum payoff structure. This method is guaranteed to converge from arbitrary initial values to the optimal value functions. However, the zero-sum payoff structure can be a very restrictive requirement in some systems and thus Hu and Wellman extended this algorithm to general-sum Markov games in [2]. Unfortunately, their method is guaranteed to converge only under very restrictive conditions. Littman proposed a new method in [3], which relaxes these limitations by adding some additional (a priori) information about the roles of the agents in the system. Wang and Sandholm proposed a method that is guaranteed to converge with any team Markov game to the optimal Nash equilibrium in [4]. Conitzer and Sandholm presented an algorithm that converges to a Nash equilibrium in self-play and learns to play optimally against stationary opponents in [5].

It still remains an open question whether any computationally efficient methods for computing Nash equilibria of finite games exist. To overcome this problem, Greenwald and Hall proposed a multiagent reinforcement learning method in [6] that uses the correlated equilibrium concept in place of the Nash equilibrium. Correlated equilibrium points can be calculated using linear programming and thus the method remains tractable also with larger problem instances. Some complexity results about Nash equilibria can be found in [7].

Our previous contributions in the field of multiagent reinforcement learning include an asymmetric multiagent reinforcement learning model [8], a hybrid model for multiagent reinforcement learning [9], and numerical methods for multiagent reinforcement learning [10, 11]. In addition, we have investigated asymmetric multiagent reinforcement learning method as a solution method for a pricing problem in [12].

We begin this paper by introducing the background and basic solution concepts of game theory. Next we discuss more complex decision making situations and compact ways to mathematically describe and solve these situations. Then we briefly go through the theory behind MDPs and introduce some practical learning methods applied to reinforcement learning. Based on the theory of MDPs, we describe multiagent systems as MGs and discuss also some solving and learning methods designed for MGs. Finally, we demonstrate the presented ideas with a small example task.

## 2. GAME THEORY

This section is mainly concerned with the basic problem settings and definitions of game theory. We start with

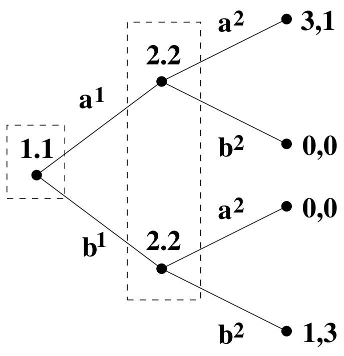

Figure 1. An example of a game in extensive form. Nodes 1.1 and 2.2 are decision nodes of the player 1 and 2, respectively. Each arch connected to a decision node (marked with $a$ and $b$) is denoting the decision of the corresponding player. Dashed boxes are information states for the corresponding player, e.g. player 2 does not observe the actual strategy choice of the player 1. The $i$th number in a leaf node is the resulting payoff for the player $i$.

some preliminary information about mathematical games and then proceed to their solution concepts which are essential for the rest of the paper. Finally, we discuss combining different solution concepts.

## 2.1. Basic concepts

Mathematical games can be defined by using different representations. The most important forms are the *extensive* form and the *strategic* form. Although the extensive form is the most richly structured way to describe game situations, the strategic form is conceptually simpler and can be derived (*normal representation*) from the extensive form. In this paper, we use games in strategic form for making decisions at each time step.

Games in strategic form are usually referred to as *matrix games* and particularly in the case of two players, if the payoff matrices for both players are separated, as *bimatrix games*. In general, an $N$-person matrix game is defined as follows:

**Definition 1** *A* matrix game *is a tuple* $\Gamma = (A^1, \dots, A^N, r^1, \dots, r^N)$*, where* $N$ *is the number of players,* $A^i$ *is the strategy space for player* $i$ *and* $r^i : A^1 \times A^2 \times \ldots \times A^N \to \mathbb{R}$ *is the payoff function for player* $i$*.*

An example extensive form game can be seen in Figure 1 and its normal representation in Table 1.

Table 1. The normal form representation of the extensive form game in Figure 1.

| | $a^2$ | $b^2$ |
|---|---|---|
| $a^1$ | $3,1$ | $0,0$ |
| $b^1$ | $0,0$ | $1,3$ |

In a matrix game, each player $i$ simultaneously implements a strategy $a^i \in A^i$. In addition to pure strategies $A^i$, we allow the possibility that the player uses a random (mixed) strategy. If we denote the space of probability distributions over a set $A$ by $\Delta(A)$, a randomization by a player over its pure strategies is denoted by $\sigma^i \in \Sigma^i \equiv \Delta(A^i)$.

## 2.2. Equilibrium concepts

In decision problems with only one decision maker, it is adequate to maximize the expected utility of the decision maker. However, in games there are many players and we need to define more elaborated solution concepts. Next we will shortly present two relevant solution concepts of matrix games.

**Definition 2** *If* $N$ *is the number of players, the strategies* $\sigma^1_*, \dots, \sigma^N_*$ *constitute a* Nash equilibrium *solution of the game if the following inequality holds for all* $\sigma^i \in \Sigma^i$ *and for all* $i$*:*

$$r^i(\sigma^1_*, \dots, \sigma^{i-1}_*, \sigma^i, \sigma^{i+1}_*, \dots, \sigma^N_*) \le r^i(\sigma^1_*, \dots, \sigma^N_*)$$

The idea of the Nash equilibrium solution is that the strategy choice of each player is a best response to its opponents' play and therefore there is no need for deviation from this equilibrium point for any player alone. Thus, the concept of Nash equilibrium solution provides a reasonable solution concept for a matrix game when the roles of the players are symmetric. Note that this case corresponds the situation depicted in Figure 1, in which player 2 does not know the action selection of player 1.

However, there are decision problems in which one of the players has the ability to enforce its strategy to other players. For solving these kind of optimization problems we have to use a hierarchical equilibrium solution concept, i.e. the *Stackelberg equilibrium* concept. In the two-player case, where one player is acting as the leader (player 1) and the another as the follower (player 2), the leader enforces its strategy to the opponent and the follower reacts rationally to this enforcement.

The basic idea is that the leader selects its strategy so that he enforces the opponent to select the response that leads to the optimal response for the leader. Algorithmically, in the case of finite bimatrix games where player 1 is the leader and player 2 is the follower, obtaining a Stackelberg solution $(a^1_S, a^2_S(a^1))$ can be seen as the following two-step algorithm:

1. $a^2_S(a^1) = \arg\max_{a^2 \in A^2} r^2(a^1, a^2)$
2. $a^1_S = \arg\max_{a^1 \in A^1} r^1(a^1, a^2_S(a^1))$

In step 1, the follower's strategy is expressed as a function of the leader's strategy. In step 2, the leader maximizes its own utility by selecting the optimal strategy pair. The only requirement is that the follower's response is unique; if this is not the case, some additional restrictions must be set. This setting can be expressed as an extensive form

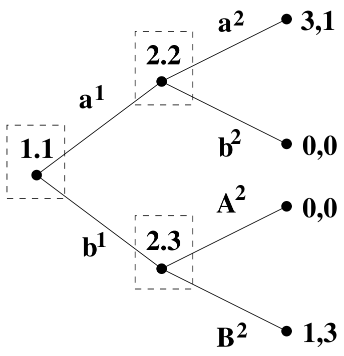

Figure 2. An example extensive form game representing two-player Stackelberg solution with player 1 acting as the leader.

Table 2. The normal form representation of the extensive form game in Figure 2.

| | $a^2A^2$ | $a^2B^2$ | $b^2A^2$ | $b^2B^2$ |
|---|---|---|---|---|
| $a^1$ | $3,1$ | $3,1$ | $0,0$ | $0,0$ |
| $b^1$ | $0,0$ | $1,3$ | $0,0$ | $1,3$ |

game depicted in Figure 2, in which player 2 knows what strategy player 1 has selected at the previous time step.

The normal form of this game is shown in Table 2. One of the Nash equilibria of this strategic form game corresponds the Stackelberg solution $(a^1, a^2)$ leading to the payoff values $(3, 1)$ for the players (all Stackelberg equilibria are also Nash equilibria). Another way to solve the game is to apply the previously declared two-step algorithm for the game shown in Table 1. This method also leads to more compact representation of the extensive form game.

## 2.3. Complex game settings

A very wide range of social interactions can be modeled as mathematical games by combining symmetric and asymmetric solution concepts. For example, let us consider the extensive form game depicted in Figure 3. In this game, player 2 can distinguish between the strategies of player 1. Player 3 cannot, however, distinguish between the strategies of player 2 and hence the overall game is a combination of symmetric and asymmetric decision tasks.

The normal representation of the game in Figure 3 can be seen as a three dimensional tensor containing $2 \times 4 \times 4 = 32$ values for each player. This kind of games usually have lots of Nash equilibria and thus some additional properties should be required from the desired equilibrium solution. In addition, it is usually possible to reduce space requirements in this way.

A natural requirement for the equilibrium solution is the Subgame Perfectness Property (SGP). A solution satisfying this property can be achieved by evaluating the game tree from the leaf nodes to the root node and replacing each subtree with the value of a subtree's Nash equilibrium. When the root node is reached, the value of the whole game is evaluated. There is a very close relationship with the Stackelberg solution concept and the SGP property; in fact a solution satisfying the SGP property can been seen as an extension to the Stackelberg equilibrium solution.

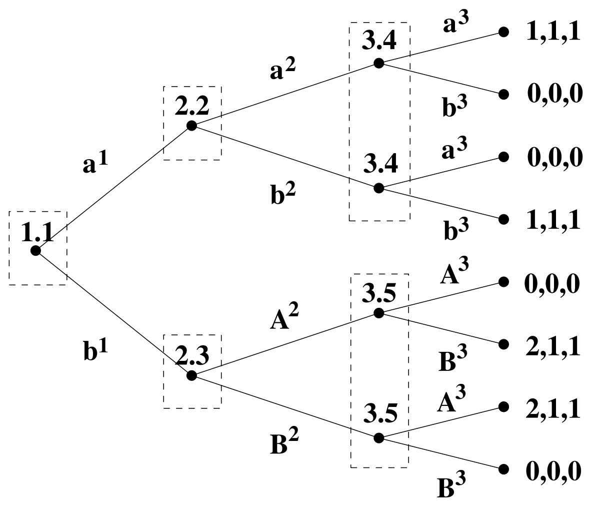

Figure 3. An example of the three-player extensive form game. Players 2 and 3 can observe the action selection of player 1 but players 2 and 3 make their decisions simultaneously.

Table 3. The resulting subgame when player 1 selects the strategy $a^1$.

| | $a^3$ | $b^3$ |
|---|---|---|
| $a^2$ | 1,1,1 | 0,0,0 |
| $b^2$ | 0,0,0 | 1,1,1 |

If we are interested only on solutions satisfying the SGP property, it is possible to reduce the actual space needed to store the game depicted in Figure 3 by keeping in mind the actual structure of the original extensive form game. Because all the players have 2 strategy options available, it suffices to store $2 \times 2 \times 2 = 8$ numbers for each player. As the players 2 and 3 know the actual strategy selection of player 1, player 1 is acting as the leader and players 2 and 3 as the followers. For making optimal decisions in the sense of SGP property, player 1 should take into account the possible responses of the other players. Tables 3 and 4 show the resulting subgames in both cases. Both subgames contain three Nash equilibria of which two are optimal in the sense that both players 2 and 3 get the maximal payoff in these equilibria. Player 1 gets a higher payoff value with the strategy selection $b^1$ than with the strategy selection $a^1$ and thus it is rational for player 1 to select strategy $b^1$.

Table 4. The resulting subgame when player 1 selects the strategy $b^1$.

| | $a^3$ | $b^3$ |
|---|---|---|
| $a^2$ | 0,0,0 | 2,1,1 |
| $b^2$ | 2,1,1 | 0,0,0 |

## 3. REINFORCEMENT LEARNING IN MARKOV DECISION PROCESSES

In this section, we briefly introduce the mathematical theory of noncompetitive *Markov decision processes*. In addition, practical solution methods for these processes are discussed at the end of this section.

### 3.1. Markov decision process

A fundamental concept in a Markov Decision Process is an *agent* that interacts with the environment in the manner illustrated in Figure 4. The environment evolves (changes its state) probabilistically and for each state there is a set of possible actions that the agent may take. Every time the agent takes an action, a certain cost is incurred.

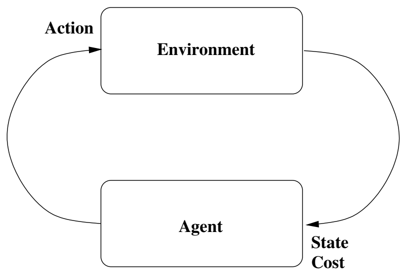

Figure 4. An overview of the learning system.

Formally, we define the Markov decision process as follows:

**Definition 3** *A* Markov Decision Process (MDP) *is a tuple* $(S, A, p, r)$*, where* $S$ *is the set of all states,* $A$ *is the set of all actions,* $p : S \times A \to \Delta(S)$ *is the state transition function and* $r : S \times A \to \mathbb{R}$ *is the reward function.* $\Delta(S)$ *is the set of probability distributions over the set* $S$*.*

Additionally, we need a *policy*, i.e. a rule stating what to do, given the knowledge of the current state of the environment. The policy is defined as a function from states to actions:

$$\pi : S_t \to A_t, \tag{1}$$

where $t$ refers to the discrete time step. The policy is *stationary* if there are no time dependents, i.e. :

$$\pi : S \to A. \tag{2}$$

In this paper, we are only interested about stationary policies. The goal of the agent is to find the policy $\pi_*$ that maximizes its expected discounted utility $R$:

$$V_\pi(s) = E_\pi[R|s_0 = s] = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|s_0 = s\right], \tag{3}$$

where $r_t$ is an immediate reward at time step $t$ and $\gamma$ is a discount factor. Moreover, the value for each state-action pair is:

$$Q_\pi(s,a) = E_\pi[R|s_0 = s, a_0 = a] = r(s,a) + \gamma\sum_{s'} p(s'|s,a)V_\pi(s'). \tag{4}$$

Finding the optimal policy $\pi_*$ can be seen as an optimization problem, which can be solved e.g. using dynamic programming algorithms.

### 3.2. Solving MDPs

Using dynamic programming requires solving the following equation for all states $s \in S$:

$$V_{\pi_*}(s) = \max_{a\in A(s)} Q_{\pi_*}(s,a). \tag{5}$$

These equations, *Bellman optimality equations*, form a basis for reinforcement learning algorithms. There are two basic methods for calculating the optimal policy, *policy iteration* and *value iteration*. In the policy iteration algorithm, the current policy is evaluated and then improved using greedy optimization based on the evaluation step. The value iteration algorithm is based on successive approximations of the value function and there is no need for repeated computation of the exact value function.

In both algorithms, the exact model of the environment should be known a priori. In many situations, however, we do not have the model available. Fortunately, it is possible to approximate the model from individual samples on-line. These methods are called temporal difference methods and can be divided to off-policy and on-policy methods based on whether they are using the same policy they are optimizing for learning or not. An example of on-policy methods is *SARSA-learning* which has the update rule [13]:

$$\begin{aligned} Q_{t+1}(s_t,a_t) &= (1-\alpha_t)Q_t(s_t,a_t) \\ &\quad + \alpha_t[r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1})], \end{aligned} \tag{6}$$

where the action selection in the state $s_{t+1}$ occurs according to the current policy. An example of off-policy methods is Q-learning. Its update rule is [14]:

$$\begin{aligned} Q_{t+1}(s_t,a_t) &= (1-\alpha_t)Q_t(s_t,a_t) \\ &\quad + \alpha_t[r_{t+1} + \gamma \max_{b\in A} Q_t(s_{t+1},b)]. \end{aligned} \tag{7}$$

## 4. MULTIAGENT REINFORCEMENT LEARNING IN MARKOV GAMES

Until now, we have only discussed the case where there is only one agent in the environment. In this section we extend the theory of MDPs to the case of multiple decision makers in the same environment. At the end of the section, a number of solving and learning methods for this extended model are briefly discussed.

### 4.1. Markov games

With multiple agents in the environment, the fundamental problem of single-agent MDPs is that the approach treats the other agents as a part of the environment and thus ignores the fact that the decisions of the other agents may influence the state of the environment.

One possible solution is to use competitive multiagent Markov decision processes, i.e. *Markov games*. In a Markov game, the process changes its state according to the action choices of all the agents and can thus be seen as a multicontroller Markov decision process. Formally, we define a Markov game as follows:

**Definition 4** *A* Markov game *(*stochastic game*) is defined as a tuple* $(S, A^1, \ldots, A^N, p, r^1, \ldots, r^N)$*, where* $N$ *is the number of agents,* $S$ *is the set of all states,* $A^i$ *is the set of all actions for each agent* $i \in \{1, N\}$*,* $p : S \times A^1 \times \ldots \times A^N \rightarrow \Delta(S)$ *is the state transition function,* $r^i : S \times A^1 \times \ldots \times A^N \rightarrow \mathbb{R}$ *is the reward function for agent* $i$*.* $\Delta(S)$ *is the set of probability distributions over the set* $S$*.*

Again, as in the case of single-agent MDP, we need a policy $\pi^i$ for each agent $i$ (the policies are assumed to be stationary):

$$\pi^i : S \rightarrow A^i, \forall i \in \{1, N\}. \tag{8}$$

The expected discounted utility of agent $i$ is the following:

$$\begin{aligned} V^i_{\pi^1,\ldots,\pi^N}(s) &= E_{\pi^1,\ldots,\pi^N}[R^i | s_0 = s] \\ &= E_{\pi^1,\ldots,\pi^N}\left[\sum_{t=0}^{\infty} \gamma^t r^i_{t+1} | s_0 = s\right], \end{aligned} \tag{9}$$

where $r^i_t$ is the immediate reward at time step $t$ for agent $i$ and $\gamma$ is a discount factor. Moreover, the value for each state-action pair is

$$\begin{aligned} &Q^i_{\pi^1,\ldots,\pi^N}(s, a^1, \ldots, a^N) \\ &= E_{\pi^1,\ldots,\pi^N}[R^i | s_0 = s, a^1_0 = a^1, \ldots, a^N_0 = a^N] \\ &= r^i(s, a^1, \ldots, a^N) \\ &+ \gamma \sum_{s'} p(s' | s, a^1, \ldots, a^N) V^i_{\pi^1,\ldots,\pi^N}(s'). \end{aligned} \tag{10}$$

Contrast to the single-agent MDP, finding the optimal policy $\pi^i_*$ for each agent $i$ can be seen as a game theoretical problem where the strategies the players can choose are the policies defined in Eq. (8).

## 4.2. Solving Markov games

In the case of multiagent reinforcement learning, it is not enough to maximize the expected utilities of individual agents. Instead, our goal is to find an equilibrium policy of the Markov game, e.g. a Nash equilibrium policy. The Nash equilibrium policy is defined as follows:

**Definition 5** *If* $N$ *is the number of agents and* $\Pi^i$ *is the policy space for agent* $i$*, the policies* $\pi^1_*, \ldots, \pi^N_*$ *constitute a Nash equilibrium solution of the game if the following inequality holds for all* $\pi^i \in \Pi^i$ *and for all* $i$ *in each state:*

$$V^i_{\pi^1_*,\ldots,\pi^i,\ldots,\pi^N_*}(s) \leq V^i_{\pi^1_*,\ldots,\pi^N_*}(s)$$

It is noteworthy that Definition 5 coincides with Definition 2 when individual strategies are replaced with policies. The Stackelberg equilibrium concept can be defined for policies in similar fashion. We refer to methods build on Markov games with the Nash equilibrium concept as symmetric methods and to methods that utilize the Stackelberg equilibrium concept as asymmetric methods.

If the exact model, i.e. rewards and state transition probabilities, is known a priori, it is possible to solve the game using standard mathematical optimization methods. However, only a few special cases of Markov games can be solved with linear programming and, in general, more complex methods are needed.

## 4.3. Symmetric learning in Markov games

As in the case of single agent reinforcement learning, Q-values defined in Eq. (10) can be learned from observations on-line using some iterative algorithm. For example, in the two-agent case, if we use Q-learning, the update rule for agent 1 is [15]:

$$\begin{aligned} Q^1_{t+1}(s_t, a^1_t, a^2_t) &= (1 - \alpha_t) Q^1_t(s_t, a^1_t, a^2_t) \\ &+ \alpha_t [r^1_{t+1} + \gamma \text{Nash}\{Q^1_t(s_{t+1})\}], \end{aligned} \tag{11}$$

where Nash$\{Q^1_t(s_{t+1})\}$ is the Nash equilibrium outcome of the bimatrix game defined by the payoff function $Q^1_t(s_{t+1})$. The update rule for agent 2 is symmetric.

Note that it is guaranteed that every finite matrix game possesses at least one Nash equilibrium in mixed strategies. However, there need not exist a Nash equilibrium point in pure strategies and therefore Nash$\{Q^1_t(s_{t+1})\}$ in Eq. (11) returns the value of a mixed strategy equilibrium.

## 4.4. Asymmetric learning in Markov games

A Markov game can be seen as a set of matrix games associated with each state $s \in S$. If the value functions of both the leader and the follower are known, we can obtain an asymmetric solution of the Markov game by solving the matrix game associated with each state $s$ using the Stackelberg solution concept. The following three stage protocol solves a Stackelberg equilibrium solution in a state $s \in S$:

1. Determination of the cooperation strategies $a^c = (a^{1c}, a^{2c})$ by finding the maximum element of the matrix game $Q^1_{\pi_1,\pi_2}$ in the state $s$:

$$\arg \max_{\substack{a^1 \in A^1 \\ a^2 \in A^2}} Q^1_{\pi^1,\pi^2}(s, a^1, a^2). \tag{12}$$

2. Determination of the leader's enforcement (and action, $a^1_S = g(s, a^c)$):

$$g(s, a^c) = \arg \min_{a^1 \in A^1} \| f(Q^2_{\pi^1,\pi^2}(s, a^1, a^2)), a^c \|. \tag{13}$$

3. Determination of the follower's response $a^2_S$:

$$a^2_S = \arg \max_{a^2 \in A^2} Q^2_{\pi^1,\pi^2}(s, g(s, a^c), a^2). \tag{14}$$

In the protocol, $\|a, a^c\|, a \in A^2$ is a distance measure, defined in the Q-value space of the leader, measuring the distance between the Q-value corresponding a particular action and the Q-value associated to the cooperation strategies (maximal possible payoff for the leader), i.e. :

$$\|x, a^c\| = |Q^1_{\pi^1,\pi^2}(s, a^1, x) - Q^1_{\pi^1,\pi^2}(s, a^{1c}, a^{2c})|. \quad (15)$$

The function $f$ is used to select actions by player 2; e.g. in the case of of greedy action selection $f = \arg\max_{a^2 \in A^2}$. In practical implementations of the protocol, e.g. when the protocol is applied to action selection during learning, the minimization in step 2 can be replaced with the *softmin* function and the maximization in step 3 with the *softmax* function for ensuring the proper exploration of the state-action space.

Actual learning of the payoffs $Q^1_{\pi^1_S,\pi^2_S}$ and $Q^2_{\pi^1_S,\pi^2_S}$ can be done by using any suitable method from the field of reinforcement learning. In this paper we present the equations for asymmetric multiagent Q-learning. If agent 1 is the leader and agent 2 is the follower, update rules for the Q-values are as follows:

$$\begin{aligned} Q^1_{t+1}(s_t, a^1_t, a^2_t) &= (1-\alpha_t)Q^1_t(s_t, a^1_t, a^2_t) \\ &+ \alpha_t[r^1_{t+1} + \gamma \max_{b \in A^1} Q^1_t(s_{t+1}, b, Tb)] \end{aligned} \quad (16)$$

and

$$\begin{aligned} Q^2_{t+1}(s_t, a^1_t, a^2_t) &= (1-\alpha_t)Q^2_t(s_t, a^1_t, a^2_t) \\ &+ \alpha_t[r^2_{t+1} \\ &+ \gamma \max_{b \in A^2} Q^2_t(s_{t+1}, g(s_{t+1}, a^c_{t+1}), b)]. \end{aligned} \quad (17)$$

In Eq. (16), the operator $Tb$ conducts the follower's response to the leader's action enforcement $b$.

### 4.5. Learning in complex decision tasks

As discussed above, various social interaction situations between agents can be learned by combining asymmetric and symmetric learning methods. It is possible to solve all problem instances by using the normal form representation in each state but the number of strategies grows very fast with the number of information states. Therefore it would be more efficient to evaluate the state values by using the techniques presented in Section 2.3.

The general form of the temporal-difference learning rule takes the following form:

$$\begin{aligned} Q^i_{t+1}(s_t, a^1_t, \ldots, a^N_t) &= (1-\alpha_t)Q^i_t(s_t, a^1_t, \ldots, a^N_t) \\ &+ \alpha_t[r^i_{t+1} + \gamma \mathrm{f}\{Q^i_t(s_{t+1})\}], \end{aligned} \quad (18)$$

where the operator f evaluates the value of a state and thus works as discussed in Section 2.3. Note that the learning rule is the same for all agents in the system but the actual implementation of the operator f may be different for different agents.

Table 5. The subgame when player 1 selects the strategy $a^1$ in state 1.

| | $a^3$ | $b^3$ |
|---|---|---|
| $a^2$ | 598.492 | 595.492 |
| $b^2$ | 595.492 | 598.492 |

Table 6. The subgame when player 1 selects the strategy $b^1$ in state 2.

| | $a^3$ | $b^3$ |
|---|---|---|
| $a^2$ | 601.508 | 592.508 |
| $b^2$ | 592.508 | 601.508 |

## 5. SIMPLE EXAMPLE

In this section, we solve a simple example task with two states and three learning agents. Each agent has two options available in both states. All options cause the system to switch its state to the other state. The example is solved by using the off-policy method defined by Eq. (18). Figure 5 illustrates the decision tasks in both states and also lists the payoff values for each agent. In state 1, an ordering among the agents is only partial (agents 2 and 3 make their decisions simultaneously) whereas in state 2 the ordering is full.

As all the agents have two options available in both states, a three dimensional array for each agent and for each state is needed for storing the Q-values during the learning process. The learned Q-values for agent 1 in the case of action selection $a^1$ in state 1 and action selection $b^1$ in state 2 are shown in Tables 5 and 6, respectively. The discount factor $\gamma$ was 0.99 in the simulation runs and all state-action tuples were visited 5000 times.

In Figure 6, the convergence of the Q-values of agent 1 is illustrated. In this figure, the Euclidean distance between vectors containing values from consecutive training rounds is plotted against the round number. Only the case of agent 1 in state 1 is shown. The system converged to the optimal value function and the changes in Q-values were very small after 700 iterations. Convergence properties in all other cases are similar.

## 6. CONCLUSIONS

In this paper, we proposed the idea of associating arbitrarily complex decision tasks with states in multiagent

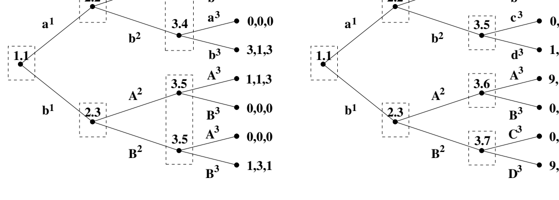

Figure 5. Extensive form games corresponding to the states in the example. In state 1 (left), an ordering among the agents is only partial whereas in state 2 (right) it is full.

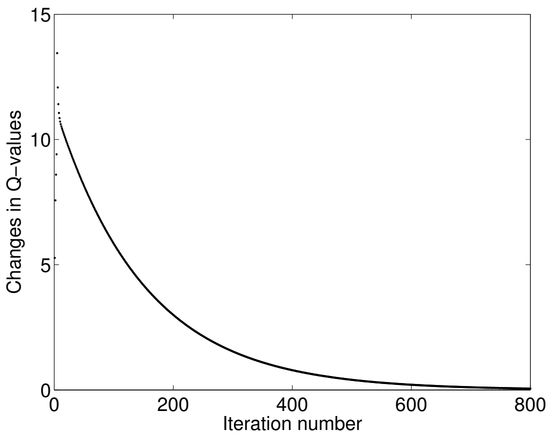

Figure 6. Convergence of Q-values in the two-state example. The learning rate parameter was a constant. Only the case of agent 1 in state 1 is plotted. Convergence curves in other cases are similar.

reinforcement learning systems. These decision tasks are represented as mathematical games and it is possible to reduce the space needed for storing these games by combining asymmetric and symmetric learning models. In addition, a concrete learning method is presented for off-policy reinforcement learning in Markov games and a simple two-state learning task is solved by using this method.

Generally, space and computational requirements increase very fast as the number of learning agents is increased. Therefore it would be an interesting future research direction to apply function approximators such as neural networks in the approximation of value functions and policies directly. Additionally, the proposed methods will be tested with larger, realworld applications.

## 7. REFERENCES

[1] Michael L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” in *Proceedings of the Eleventh International Conference on Machine Learning (ICML-1994)*, New Brunswick, NJ, 1994, pp. 157–163.

[2] Junling Hu and Michael P. Wellman, “Multiagent reinforcement learning: Theoretical framework and an algorithm,” in *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-1998)*, Madison, WI, 1998, pp. 242–250.

[3] Michael L. Littman, “Friend-or-Foe Q-learning in general-sum games,” in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, Williamstown, MA, 2001, pp. 322–328.

[4] Xiaofeng Wang and Tuomas W. Sandholm, “Reinforcement learning to play an optimal Nash equilibrium in team Markov games,” in *Advances in Neural Information Processing Systems (NIPS-2002)*, Vancouver, British Columbia, Canada, 2003, pp. 1603–1610.

[5] Vincent Conitzer and Tuomas W. Sandholm, “AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents,” in *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington, DC, 2003, pp. 83–90.

[6] Amy Greenwald and Keith Hall, “Correlated-Q learning,” in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2003)*, Washington, DC, 2003, pp. 242–249.

[7] Vincent Conitzer and Tuomas W. Sandholm, “Complexity results about Nash equilibria,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2003)*, Acapulco, Mexico, 2003, pp. 765–771.

[8] Ville J. Könönen, “Asymmetric multiagent reinforcement learning,” *Web Intelligence and Agent Systems: An International Journal (WIAS)*, vol. 2, no. 2, pp. 105–121, 2004.

[9] Ville J. Könönen, “Hybrid model for multiagent reinforcement learning,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN-2004)*, Budapest, Hungary, 2004, pp. 1793–1798.

[10] Ville J. Könönen, “Gradient based method for symmetric and asymmetric multiagent reinforcement learning,” in *Proceedings of the Fourth International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-2003)*, Hong Kong, China, 2003, pp. 68–75.

[11] Ville J. Könönen, “Policy gradient method for multiagent reinforcement learning,” in *Proceedings of the second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-2003)*, Singapore, 2003, CD-ROM.

[12] Ville J. Könönen and Erkki Oja, “Asymmetric multiagent reinforcement learning in pricing applications,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN-2004)*, Budapest, Hungary, 2004, pp. 1097–1102.

[13] Gavin A. Rummery and Mahesan Niranjan, “On-line Q-learning using connectionist systems,” Tech. Rep. CUED/F-INFENG/TR166, Cambridge University, Engineering Department, 1994.

[14] Christopher J.C.H Watkins, *Learning from Delayed Rewards*, Ph.D. thesis, Cambridge University, 1989.

[15] Junling Hu and Michael P. Wellman, “Nash Q-learning for general-sum stochastic games,” *Journal of Machine Learning Research*, vol. 4, pp. 1039–1069, 2003.