Multiagent Reinforcement Learning With Unshared Value Functions

Yujing Hu, Yang Gao, *Member, IEEE,* and Bo An, *Member, IEEE*

***Abstract*—One important approach of multiagent reinforcement learning (MARL) is equilibrium-based MARL, which is a combination of reinforcement learning and game theory. Most existing algorithms involve computationally expensive calculation of mixed strategy equilibria and require agents to replicate the other agents' value functions for equilibrium computing in each state. This is unrealistic since agents may not be willing to share such information due to privacy or safety concerns. This paper aims to develop novel and efficient MARL algorithms without the need for agents to share value functions. First, we adopt pure strategy equilibrium solution concepts instead of mixed strategy equilibria given that a mixed strategy equilibrium is often computationally expensive. In this paper, three types of pure strategy profiles are utilized as equilibrium solution concepts: pure strategy Nash equilibrium, equilibrium-dominating strategy profile, and nonstrict equilibrium-dominating strategy profile. The latter two solution concepts are strategy profiles from which agents can gain higher payoffs than one or more pure strategy Nash equilibria. Theoretical analysis shows that these strategy profiles are symmetric meta equilibria. Second, we propose a multistep negotiation process for finding pure strategy equilibria since value functions are not shared among agents. By putting these together, we propose a novel MARL algorithm called negotiation-based Q-learning (NegoQ). Experiments are first conducted in grid-world games, which are widely used to evaluate MARL algorithms. In these games, NegoQ learns equilibrium policies and runs significantly faster than existing MARL algorithms (correlated Q-learning and Nash Q-learning). Surprisingly, we find that NegoQ also performs well in team Markov games such as pursuit games, as compared with team-task-oriented MARL algorithms (such as friend Q-learning and distributed Q-learning).**

***Index Terms*—Game theory, multiagent reinforcement learning, Nash equilibrium, negotiation.**

Manuscript received December 12, 2013; revised April 22, 2014 and June 15, 2014; accepted June 17, 2014. Date of publication July 2, 2014; date of current version March 13, 2015. This work was supported in part by the National Science Foundation of China under Grant 61035003, Grant 61175042, Grant 61321491, and Grant 61202212, in part by the 973 Program of Jiangsu, China under Grant BK2011005, in part by the Program for New Century Excellent Talents in University under Grant NCET-10-0476, in part by the Program for Research and Innovation of Graduate Students in General Colleges and Universities, Jiangsu under Grant CXLX13_049, and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization. This paper was recommended by Associate Editor T. Vasilakos.

Y. Hu and Y. Gao are with the State Key Laboratory for Novel Software Technology, Department of Computer Science, Nanjing University, Nanjing 210023, China (e-mail: huyujing.yujing.hu@gmail.com; gaoy@nju.edu.cn).

B. An is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: boan@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2014.2332042

I. Introduction

A multiagent system (MAS) is defined as a group of autonomous agents that sense and interact with an environment and act for predefined goals. It brings a new paradigm to design artificial intelligence applications such as robotic teams [1], manufacturing control [2], networks [3]–[5], and so on. Due to the dynamics and complexity of environments, some machine learning techniques have been developed for MASs in order to improve the performance of each agent and the whole system.

One common learning technique in a MAS is multiagent reinforcement learning (MARL), which is an extension of reinforcement learning (RL) [6] in a multiagent domain. It has been successfully applied in real practice, such as QoS routing in ATM and broadband networks [7], [8]. A survey [9] claims that MARL can be treated as a fusion of temporal-difference RL, game theory, and direct policy search techniques. More recently, some other MARL techniques have also been proposed, including learning automata [10]–[12], and dynamic programming on feasible-sets [13].

Equilibrium-based MARL is one of the most important approaches in MARL. It adopts Markov games as the framework and introduces various equilibrium solution concepts in game theory to define optimal policies. Minimax-Q [14] is widely considered as the first equilibrium-based MARL algorithm, which uses a minimax rule to help agents select actions. Although minimax-Q only solves two-agent zero-sum Markov games, it presents a novel framework for MARL. Hu and Wellman [15], [16] proposed Nash Q-learning (NashQ) for general-sum Markov games based on the concept of Nash equilibrium. The convergence condition of NashQ requires each state's one-shot game to have a global optimal equilibrium point or a saddle point. Littman [17] proposes friend-or-foe Q-learning (FFQ) for learning two special kinds of Nash equilibria (adversarial equilibrium and coordination equilibrium) and it has less strict convergence condition compared with NashQ. Correlated Q-learning (CE-Q) [18] is proposed based on the concept of correlated equilibrium. It generalizes NashQ and allows for dependencies among agents' actions.

Unfortunately, most existing equilibrium-based MARL algorithms require agents to replicate the other agents' value functions [9] and adopt mixed strategy equilibria, which prevents them from being widely applied. The reasons are as follows. First, sharing value functions is unrealistic in domains where agents are distributed or locally restricted (e.g., grid computing [19], distributed sensor networks [20], distributed

streaming processing systems [21]). Moreover, value function is private information of an agent and the agent may not be willing to share such sensitive information with others. Second, mixed strategy equilibria are computationally expensive. Motivated by this, the goal of this paper is to develop novel and efficient MARL algorithms without sharing value functions between agents.

The proposed MARL algorithm for addressing the above two limitations is based on two key ideas. Firstly, instead of mixed strategy profiles, three types of pure strategy profiles are utilized as the equilibrium concepts in MARL: pure strategy Nash equilibrium (PNE), equilibrium-dominating strategy profile (EDSP), and nonstrict equilibrium-dominating strategy profile (nonstrict EDSP). The latter two are strategy profiles that lead to higher payoffs than one or more pure strategy Nash equilibria and encourage cooperation between agents. We prove that the three identified strategy profiles are symmetric meta equilibria. Secondly, since agents cannot replicate the other agents' value functions, a multistep negotiation process is proposed to find the three strategy profiles, through which agents can exchange their action preferences. By integrating the two ideas, a novel MARL algorithm called negotiation-based Q-learning (NegoQ) is proposed.

Experiments are first conducted in grid-world games, in which NegoQ exhibits empirical convergence and runs much faster than the state-of-the-art algorithms. Surprisingly, we also find that NegoQ can do well in team Markov games, though it is not intentionally designed for such type of games. We show that in a team Markov game called a pursuit game, NegoQ achieves equivalent (and sometimes better) performance compared with some team-task-oriented MARL algorithms (joint action learner [JAL] [22], friend Q-learning [17], distributed Q-learning [23]).

The rest of this paper is organized as follows. In the next section, background information about MARL is introduced. In Section III, three types of pure strategy profiles are defined and corresponding theoretical analysis is provided. Section IV introduces the multistep negotiation process for finding these strategy profiles, after which the algorithm NegoQ is proposed. Experimental results are shown in Sections V and VI. Section VII discusses the related work. Finally, we draw some conclusions in Section VIII. Appendix A briefly introduces metagame theory and Appendix B shows the proofs of some formal results.

II. Multiagent Reinforcement Learning

In multiagent settings, learning is complicated due to the fact that multiple agents learn simultaneously. This can thus make the environment nonstationary for a learner [24]. Game theory provides us with a powerful tool for modeling multiagent interactions. In this section, we review some basic concepts in MARL and game theory.

A. Preliminaries

Markov games are commonly adopted as the framework of MARL [14], [15], [17], [18].

Definition 1 (Markov Game): An n-agent ($n \geq 2$) Markov game is a tuple $\langle N, S, \{A_i\}_{i=1}^n, \{R_i\}_{i=1}^n, T\rangle$, where N is the set of agents; S stands for the state space of the environment; A_i is the action space of agent $i (i = 1, \ldots, n)$. Let $A = A_1 \times \cdots \times A_n$ be the joint action space of the agents. $R_i : S \times A \to \Re$ is the reward function of agent i; $T : S \times A \times S \to [0, 1]$ is the transition function of the environment.

In general, an agent i's goal is to find a policy which maximizes its expected (discounted) sum of rewards for each state s at each time step t

$$E\Big\{\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s\Big\} \tag{1}$$

where E stands for expectation, k is future time step, $\gamma \in [0, 1)$ is the discount rate, and r_i^{t+k} is agent i's immediate reward at future time step $(t+k)$. However, in MARL, one agent i cannot maximize $V_i(s)$ individually. The other agents' policies should be taken into consideration. A policy for agent i is a function $\pi_i : S \times A_i \to [0, 1]$, which assigns each state a probability distribution over the agent's action set. Denote the joint policy of all agents by $\pi = (\pi_1, \ldots, \pi_n)$. The state-value function V_i^π gives expected values for agent i under joint policy π at every state s

$$V_i^\pi(s) = E_\pi\Big\{\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s\Big\}. \tag{2}$$

The action-value function Q_i^π gives expected values if we start at state s performing a joint action $\vec{a}$ and then follow the joint policy π:

$$Q_i^\pi(s, \vec{a}) = E_\pi\Big\{\sum_{k=0}^{\infty}\gamma^k r_i^{t+k}|s_t = s, \vec{a}_t = \vec{a}\Big\}. \tag{3}$$

RL can be viewed as a sampling method which estimates the value functions V_i^π or Q_i^π of the current policy π and improves π by updating the value functions. Usually, we use V_i and Q_i to denote the value functions of agent i maintained by an algorithm. A common way for learning in a Markov game is to construct a normal-form game (a.k.a. one-shot game) in each sampled state s, treat the state-action value $Q_i(s, \vec{a})$ as each agent i's payoff of each joint action $\vec{a}$ in this game, compute an equilibrium, and update the action-value function Q_i according to that equilibrium [14], [15], [17], [18].

Definition 2 (Normal-Form Game): An n-agent ($n \geq 2$) normal-form game Γ is a tuple $\langle N, \{A_i\}_{i=1}^n, \{U_i\}_{i=1}^n\rangle$, where N is a finite set of agents and A_i is the finite action space of agent i. Let A be the joint action space and $U_i{:}A \to \Re$ be the utility function of agent i. For agent i, a joint action $\vec{a} \in A$ can be also denoted by $(a_i, \vec{a}_{-i})$, where a_i represents the action taken by agent i and $\vec{a}_{-i} \in A_{-i}$ is the joint action of all other agents.

For a MARL algorithm, let $Q_i(s) = \{Q_i(s, \vec{a})|\forall\ \vec{a}\}$ be the set of all state-action values of agent i in state s. The tuple $\langle N, \{A_i\}_{i=1}^n, \{Q_i(s)\}_{i=1}^n\rangle$ can be treated as the normal-form game played in state s, where $Q_i(s)$ corresponds to agent i's utility function U_i in Definition 2.

B. Equilibrium-Based MARL

In order to define the concept of optimal policy in a MAS, some equilibrium solution concepts in game theory are introduced into MARL.

1) Minimax-Q: Littman [14] proposed the first equilibrium-based MARL algorithm called minimax-Q, which focuses on two-agent zero-sum Markov games. The framework of minimax-Q follows that of Q-learning [25]. The form of the action-value function is $Q(s, a, o)$, which takes the agent's action a and the opponent's action o into consideration. The value of any state s is defined as

$$V(s) = \max_{\pi} \min_{o} \sum_{a} Q(s, a, o)\pi(s, a) \tag{4}$$

where π denotes any policy of the agent (i.e., $\pi(s, a)$ is the probability of taking action a in state s). The updating rule for Q-value after an experience (s, a, o, r, s') is

$$Q(s, a, o) \leftarrow (1 - \alpha)Q(s, a, o) + \alpha[r + \gamma V(s')] \tag{5}$$

where $\alpha \in [0, 1]$ is the learning rate, r is the reward received by the agent and s' is the next state.

2) NashQ: Hu and Wellman [15], [16] proposed Nash Q-learning (NashQ) algorithm for more general problems based on the concept of Nash equilibrium.

Definition 3 (Nash Equilibrium, NE): In an n-agent ($n \geq 2$) normal-form game Γ, a strategy profile $p^* = (p_1^*, \ldots, p_n^*)$, where $p_i^*: A_i \rightarrow [0, 1]$ is a probability distribution over A_i, is a Nash equilibrium if and only if for any $i \in N$, it holds that

$$U_i(p^*) \geq \max_{p_i' \in \Sigma_i} U_i(p_1^*, \ldots, p_{i-1}^*, p_i', p_{i+1}^*, \ldots, p_n^*) \tag{6}$$

where Σ_i is the strategy space of agent i. For any given strategy profile p, $U_i(p) = \Sigma_{\vec{a} \in A} p(\vec{a}) U_i(\vec{a})$.

Like minimax-Q, NashQ still adopts the framework of Q-learning, but it needs to maintain the action-value function $Q_i (i = 1, \ldots, n)$ for each agent. The rule of updating Q_i is

$$Q_i(s, \vec{a}) \leftarrow (1 - \alpha)Q_i(s, \vec{a}) + \alpha[r_i + \gamma NashQ_i(s')] \tag{7}$$

where $\vec{a}$ represents a joint action taken by all agents, and $NashQ_i(s')$ is the value of agent i in regard to the selected Nash equilibrium in state s'.

3) FFQ: Littman [17] proposed FFQ for two special types of Nash equilibria: adversarial equilibrium and coordination equilibrium, which correspond to two opponent types "friend" and "foe," respectively. For simplicity, we introduce the two-agent version of FFQ. From the perspective of agent 1, the rule of updating value function is (7) with

$$NashQ_1(s) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1(s, a_1, a_2) \tag{8}$$

if agent 2 is considered a friend. If agent 2 is considered as a foe, the rule is (7) with

$$NashQ_1(s) = \max_{\pi_1} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} Q_1(s, a_1, a_2)\pi_1(s, a_1). \tag{9}$$

Algorithm 1: General Framework of Equilibrium-Based MARL

Input: The agent set N, state space S, and joint action space A of a Markov game, learning rate α, discount rate γ, exploration factor ϵ

1 Initialization. $\forall s \in S, \forall i \in N, \forall \vec{a} \in A, Q_i(s, \vec{a}) \leftarrow 0$;
2 **foreach** *episode* **do**
3 Initialize state s;
4 **repeat**
5 $\vec{a} \leftarrow \Theta\langle Q_1(s), \ldots, Q_n(s)\rangle$ with ϵ-greedy policy;
   /* Θ is for computing an equilibrium */
6 Receive the experience $(s, \vec{a}, r, s')$;
7 $Q_i(s, \vec{a}) \leftarrow (1 - \alpha)Q_i(s, \vec{a}) + \alpha(r + \gamma\Phi_i(s'))$;
   /* Φ_i is the expected value of the equilibrium in the game occurring in state s' for agent i */
8 $s \leftarrow s'$;
9 **until** *s is a terminal state*;

4) Correlated-Q: Greenwald and Hall [18] adopted the concept of correlated equilibrium and presented the correlated-Q learning (CE-Q) algorithm. Compared to a Nash equilibrium, a correlated equilibrium [26] allows for dependencies among agents' actions. The updating rule of Q-value is

$$Q_i(s, \vec{a}) \leftarrow (1 - \alpha)Q_i(s, \vec{a}) + \alpha[r_i + \gamma CEQ_i(s')] \tag{10}$$

where $CEQ_i(s')$ is the value of agent i with respect to the selected correlated equilibrium in state s'.

5) General Framework: In addition to the above algorithms, there are also other equilibrium-based MARL algorithms, such as asymmetric Q-learning [27], optimal adaptive learning [28], etc. In general, equilibrium-based MARL algorithms can be summarized into Algorithm 1. At line 5, $\langle Q_1(s), \ldots, Q_n(s)\rangle$ denotes the normal-form game played in state s (the agent set and action space are omitted), $\Theta\langle Q_1(s), \ldots, Q_n(s)\rangle$ represents computing an equilibrium of this normal-form game, and ϵ-greedy policy means that agents choose a joint action according to the computed equilibrium with probability $(1 - \epsilon)$ and choose a random action with probability ϵ. At line 7, $\Phi_i(s')$ is the expected value of the equilibrium in state s' for agent i.

As mentioned in Section I, in most existing algorithms, mixed strategy equilibria are often adopted as the solution concepts and agents need to replicate the other agents' value functions for equilibrium computing. This prevents them from being used widely for two reasons. Firstly, agents may not be willing to sharing value functions due to safety or privacy concerns. Secondly, mixed strategy equilibria involve computationally expensive calculation. To address these problems, in the next two sections, we define novel equilibrium solution concepts and propose algorithms for finding them without disclosing agents' value functions.

(A,B)	Confess	Deny
Confess	(-9,-9)	(0,-10)
Deny	(-10,0)	(-1,-1)

(a)

A ↓	Confess	Deny
Confess	-9	0
Deny	-10	-1

(b)

B →	Confess	Deny
Confess	-9	-10
Deny	0	-1

(c)

Fig. 1. Two-agent prisoners' dilemma. (a) Game matrix. (b) What agent A observes. (c) What agent B observes.

(A,B)	b_1	b_2	b_3
a_1	**(20,40)**	(4,22)	**(29,30)**
a_2	(18,9)	**(36,19)**	(7,4)
a_3	(17,26)	(15,38)	**(27,38)**

(a)

A ↓	b_1	b_2	b_3
a_1	20	4	29
a_2	18	36	7
a_3	17	15	27

(b)

B →	b_1	b_2	b_3
a_1	40	22	30
a_2	9	19	4
a_3	26	38	38

(c)

Fig. 2. Normal-form game that shows an example of nonstrict EDSPs, where (a_1, b_1) and (a_2, b_2) are pure strategy NE while (a_1, b_3) and (a_3, b_3) are nonstrict EDSPs. (a) Game matrix. (b) What agent A observes. (c) What agent B observes.

III. Equilibrium Solution Concepts and Analysis

In this section, we introduce three kinds of pure strategy profiles and adopt them as the equilibrium solution concepts for the normal-form game played in each state of a Markov game. The three strategy profiles are PNE, EDSP, and nonstrict EDSP.[1]

A. Equilibrium Solution Concepts

Since we focus on pure strategy equilibria, one obvious choice is PNE, in which each agent's action is a best response to the other agents' joint action. However, there are two problems for a PNE. The first is that sometimes it may not exist in a game. The second problem is that it may be Pareto dominated by a strategy profile which is not a PNE. A well-known example is the prisoners' dilemma, which is shown in Fig. 1. The unique PNE of this particular game is (Confess,Confess). However, if the agents A and B choose Deny at the same time, both of them can obtain a higher payoff. In fact, (Deny,Deny) is Pareto optimal while the unique Nash equilibrium is not. An NE is based on the assumption of full rationality, but it may not be the best solution if actions are selected according to other criteria (e.g., agents are bounded rational). If A and B only know their own utilities, (Deny,Deny) must be more attractive than (Confess,Confess) to both of them. We define a strategy profile which Pareto dominates an NE as EDSP.

Definition 4 (Equilibrium-Dominating Strategy Profile, EDSP): In an n-agent ($n \geq 2$) normal-form game Γ, a joint action $\vec{a} \in A$ is an EDSP if there exists a PNE $\vec{e} \in A$ such that

$$U_i(\vec{a}) \geq U_i(\vec{e}), i = 1, 2, \ldots, n. \tag{11}$$

The definition above indicates that each agent following an EDSP can obtain a payoff no less than that of a PNE. In addition to EDSP, we also observed another kind of interesting strategy profiles, which can be found in Fig. 2. It is easy to verify that (a_1, b_1) and (a_2, b_2) are two pure strategy Nash equilibria. Now examine the two nonequilibrium strategy profiles (a_1, b_3) and (a_3, b_3). Both of them provide a higher payoff to agent A than (a_1, b_1) and a higher payoff to agent B than (a_2, b_2). Again, if we look at this game from each agent's own perspective, it can be found that the priority orders of (a_1, b_3) and (a_3, b_3) must be higher than (a_1, b_1) for A and (a_2, b_2) for B. For each agent, these two nonequilibrium strategy profiles partially Pareto dominate one certain Nash equilibrium. We call each of them a nonstrict EDSP.

[1]We call the defined strategy profiles equilibrium solution concepts since we have proven that they are symmetric meta equilibria.

Definition 5 (Nonstrict Equilibrium-Dominating Strategy Profile): In an n-agent ($n \geq 2$) normal-form game Γ, a joint action $\vec{a} \in A$ is a nonstrict EDSP if $\forall i \in N$, there exists a PNE $\vec{e_i} \in A$ such that

$$U_i(\vec{a}) \geq U_i(\vec{e_i}). \tag{12}$$

The index i of $\vec{e_i}$ indicates this equilibrium is for agent i and may be different from those of the other agents. Obviously, an EDSP is a special case of nonstrict EDSP.

In this paper, we choose the three strategy profiles defined above as the equilibrium solution concepts for learning in a Markov game. Pure strategy Nash equilibria are chosen since we focus on pure strategy equilibria. EDSP and nonstrict EDSP are chosen since they provide higher payoffs than some Nash equilibria to agents. Rather than requiring agents to be infinite rational, the two strategy profiles allow for cooperation between agents.

B. Theoretical Analysis

In this subsection, we show that the three types of strategy profiles are symmetric meta equilibria [29]. In order to keep contextual coherence, the introduction of metagame theory and the proofs of our formal results are put in Appendix.

Theorem 1: Any PNE of a normal-form game is also a symmetric meta equilibrium.

Theorem 2: Any nonstrict EDSP of a normal-form game is a symmetric meta equilibrium.

Obviously, an EDSP is also a symmetric meta equilibrium since it is a special case of nonstrict EDSP. In short, a joint action $\vec{a}$ of a normal-form game is a symmetric meta equilibrium if for any agent i there holds

$$U_i(\vec{a}) \geq \min_{\vec{a_{-i}} \in A_{-i}} \max_{a_i \in A_i} U_i(a_i, \vec{a_{-i}}) \tag{13}$$

where U_i is the utility function of i. Symmetric meta equilibrium is a special case of meta equilibrium and it can at least satisfy each agent's optimistic expectations (the minimum value among the best response values to the other agents's joint actions). The relationship between all these equilibria can be illustrated by Fig. 3.

However, since all the three defined strategy profiles are related to PNE, one potential problem is that they may not exist in a game. In this situation, an alternative solution is

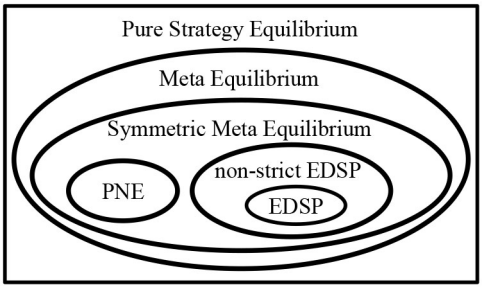


Fig. 3. Relationship between meta equilibrium, symmetric meta equilibrium, PNE, EDSP, and nonstrict EDSP. They are pure strategy equilibria.

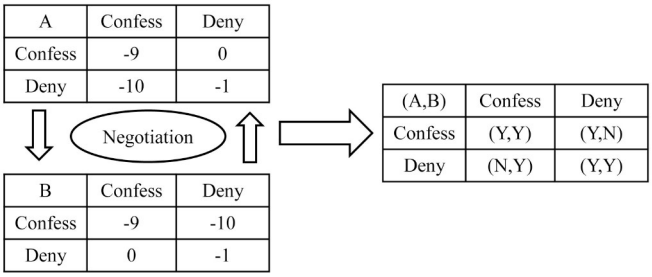


Fig. 4. Example to illustrate negotiation. Agents A and B can exchange their action preferences, after which a preference matrix can be obtained.

needed for replacing them. Fortunately, the following theorem provides a possible solution.

Theorem 3: For any normal-form game defined in Definition 2, it has at least one meta equilibrium.

Our idea is to choose a meta equilibrium when there is no pure strategy Nash equilibria and nonstrict EDSPs in a game (shown in Appendix). In the next section, we try to deal with the problem of how to obtain these strategy profiles in a distributed manner and then propose a novel MARL algorithm.

IV. Negotiation-Based Q-Learning

Since agents cannot replicate the other agents' value functions, it is impossible to compute an equilibrium according to the game matrix of each state. An alternative way should be found to obtain the strategy profiles identified in the last section. In this section, we provide a multistep negotiation process for finding the three strategy profiles without disclosing each agent's value function in MARL.

Our idea is that agents can exchange preferences of joint actions. Each agent asks the other agents whether a joint action is acceptable or not. At the same time, each agent answers similar questions received from the other agents. This idea is illustrated by Fig. 4, where a preference matrix can be obtained through negotiation between the two agents A and B. In the preference matrix, "Y" and "N" stand for an agent's positive and negative attitude to a joint action, respectively. A joint action is acceptable only when both the agents would like to choose it.

A. Negotiation Process

The negotiation process contains three steps: 1) negotiation for finding the set of pure strategy NE; 2) negotiation for finding the set of nonstrict EDSPs; and 3) negotiation for choosing one equilibrium from the sets obtained in the first two steps. Computing the set of EDSPs is not necessary since an EDSP is a special case of nonstrict EDSP. As we have mentioned before, if both the PNE and nonstrict EDSP sets are empty, a set of meta equilibria will be found through negotiation as the alternative solution, which is always nonempty. The negotiation for meta equilibrium set will be shown in Appendix.

Algorithm 2: Negotiation for Pure Strategy NE Set

Input: A normal-form game $\langle N, \{A_i\}_{i=1}^n, \{U_i\}_{i=1}^n \rangle$

/* But agent i only knows N, $\{A_i\}_{i=1}^n$, and U_i. */

The set of pure strategy Nash equilibria $J_{NE} \leftarrow \emptyset$;

Compute the maximal utility set $MaxSet_i$;

foreach $\vec{a_{-i}} \in A_{-i}$ **do**

  $a_i \leftarrow \arg\max_{a_i'} U_i(a_i', \vec{a_{-i}})$;

  /* There may be more than one such actions. */

  $J_{NE} \leftarrow J_{NE} \cup \{(a_i, \vec{a_{-i}})\}$;

/* QUESTIONING THREAD: */

foreach *joint action* $\vec{a} \in J_{NE}$ **do**

  Ask all the other agents whether $\vec{a}$ is also in their J_{NE} sets;

  if *one of the answers are 'no'* **then**

    $J_{NE} \leftarrow J_{NE} \setminus \{\vec{a}\}$;

    Tell the other agents to remove $\vec{a}$ from their J_{NE} sets;

/* ANSWERING THREAD: */

foreach *joint action* $\vec{a}'$ *received from the other agents* **do**

  if $\vec{a}'$ *is in* $MaxSet_i$ **then**

    Send answer 'yes' back to the agent;

  else

    Send answer 'no' back to the agent;

1) Negotiation for the Set of Pure Strategy Nash Equilibria: Negotiation process for finding the set of pure strategy Nash equilibria is shown in Algorithm 2, which is described from the viewpoint of agent i. The first step is to find the set of strategy profiles that are potentially Nash equilibria according to agent i's own utilities. The second step is to filter out all strategy profiles that are not Nash equilibria by asking other agents whether the strategy profiles in the set J_{NE} are also in their J_{NE} sets. At the same time, agent i also answers similar questions received from the other agents.

2) Negotiation for the Set of NonStrict EDSP: Similar to the first step, each agent can first compute potential nonstrict EDSPs according to its own utilities and then obtain the intersection through negotiation. The algorithm is shown in Algorithm 3.

3) Negotiation for Choosing One Equilibrium: Last step is to choose one equilibrium from the obtained sets for the

Algorithm 3: Negotiation for Nonstrict EDSPs Set

Input: A normal-form game $\langle N, \{A_i\}_{i=1}^n, \{U_i\}_{i=1}^n \rangle$, the set of pure strategy Nash equilibria J_{NE}

/* But agent i only knows N, $\{A_i\}_{i=1}^n$, and U_i. */

The set of nonstrict EDSPs $J_{NS} \leftarrow \emptyset$;

$X \leftarrow A \setminus J_{NE}$;

/* A is the joint action space */

foreach *Nash equilibrium* $\vec{e} \in J_{NE}$ **do**

  foreach *joint action* $\vec{a} \in X$ **do**

    if $U_i(\vec{a}) \geq U_i(\vec{e})$ **then**

      $X \leftarrow X \setminus \{\vec{a}\}$;

      $J_{NS} \leftarrow J_{NS} \cup \{\vec{a}\}$;

/* QUESTIONING THREAD: */

foreach *joint action* $\vec{a} \in J_{NS}$ **do**

  Ask all the other agents whether $\vec{a}$ is also in their J_{NS} sets;

  if *one of the answers is 'no'* **then**

    $J_{NS} \leftarrow J_{NS} \setminus \{\vec{a}\}$;

/* ANSWERING THREAD: */

foreach *joint action* $\vec{a}'$ *received from the other agents* **do**

  if $\vec{a}'$ *is in* J_{NS} **then**

    Send 'yes' back to the agent;

  else

    Send 'no' back to the agent;

Algorithm 4: Negotiation-Based Q-Learning Algorithm

Input: The agent set N, state space S, and joint action space A of a Markov game, learning rate α, discount rate γ, exploration factor ϵ

Initialization. $\forall s \in S$ and $\forall\ \vec{a} \in A$, $Q_i(s, \vec{a}) \leftarrow 0$;

foreach *episode* **do**

  Initialize state s;

  Negotiate with the other agents according to $Q_i(s)$ using Algorithms 2, 3;

  $\vec{a} \leftarrow$ the selected equilibrium (with ϵ-greedy);

  repeat

    Receive the experience $(s, \vec{a}, r_i, s')$;

    Negotiate with the other agents according to $Q_i(s')$ using Algorithms 2, 3;

    $\vec{a}' \leftarrow$ the selected equilibrium (with ϵ-greedy);

    $Q_i(s, \vec{a}) \leftarrow (1-\alpha) Q_i(s, \vec{a}) + \alpha(r_i + \gamma Q_i(s, \vec{a}'))$;

    $s \leftarrow s'$, $\vec{a} \leftarrow \vec{a}'$;

  until *s is a terminal state*;

agents. We use a simple rule here. That is, each agent first sends its favorite equilibrium (i.e., the equilibrium with the highest utility) to the other agents and then the earliest sent equilibrium will be chosen. We assume that each agent sends its request at different time, since in real practice, it is difficult for two agents to send their requests precisely at the same time. However, if two or more agents do simultaneously send their requests, there are other mechanisms which can address this problem. For example, we can choose one agent as the leader agent before learning and this agent can randomly choose one strategy profile from the sets obtained in the first two negotiation steps.

B. Complexity of Negotiation

For an n-agent normal-form game Γ, let $|J_{NE}|$ denote the number of pure strategy Nash equilibria. In Algorithm 2, the cost for each agent to compute its potential set of pure strategy Nash equilibria set J_{NE} is $O(\Pi_{i=1}^n |A_i|)$ while the total communication cost of all agents is $O((n-1)\Pi_{i=1}^n |A_i|)$ in the worst case, where each joint action of the game is a PNE. In Algorithm 3, the computational cost is $O(|J_{NE}| \times (\Pi_{i=1}^n |A_i| - |J_{NE}|))$ at most and the total communication cost for finding the set of all nonstrict EDSPs is $O((n-1)(\Pi_{i=1}^n |A_i| - |J_{NE}|))$ in the worst case, where each nonequilibrium joint action is a nonstrict EDSP.

While the complexity of these negotiation algorithms seems high, the communication cost can be reduced in some simple ways. For example, if agent i receives a joint action from agent j and it is also in agent i's J_{NE} set, there is no need for agent i to ask agent j the same question. Furthermore, an agent can broadcast a message to the other agents after it finds an NE or EDSP. In our experiments (shown in the next two sections), it only takes a few microseconds (4–30 μs) for our negotiation algorithms to find an equilibrium in two-agent 4×4 and three-agent $4 \times 4 \times 4$ normal-form games, while for computing a correlated equilibrium or a Nash equilibrium, it takes the corresponding algorithms a few hundreds microseconds or several milliseconds (129μs–15ms).

C. NegoQ

Based on the negotiation process above, we present a novel MARL algorithm called the negotiation-based Q-learning (NegoQ). As shown in Algorithm 4, NegoQ is described from the viewpoint of agent i. Its main structure follows the framework of equilibrium-based MARL in Algorithm 1. In each state s, an equilibrium is obtained and agent i takes an action according to this equilibrium with ϵ-greedy policy. After that, the reward r_i is received and the next state s' is observed, followed by the step of updating the Q-table Q_i. Agent i follows the same process iteratively until a terminal state is reached.

The major difference between NegoQ and previous algorithms is that the solution concepts adopted in NegoQ are pure strategy profiles and agents only need to keep their own value functions. On one hand, a decentralized algorithm is necessary since in some domains agents are distributed or locally restricted, and not sharing agents' value functions can reduce the memory cost and guarantee safety/privacy. On the other hand, the choice of pure strategy equilibrium solution concepts makes it possible to quickly find them without disclosing their payoffs.

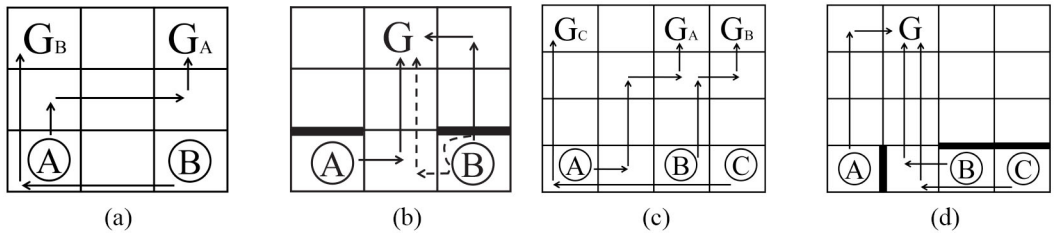


Fig. 5. Initial states and goals of agents in the four games. The arrow lines show a pure strategy equilibrium policy in each grid world. The dashed arrows in GW2 represents that if agent B is obstructed by a barrier in the first step, it will choose to move left in its second step. (a) GW1. (b) GW2. (c) GW3. (d) GW4.

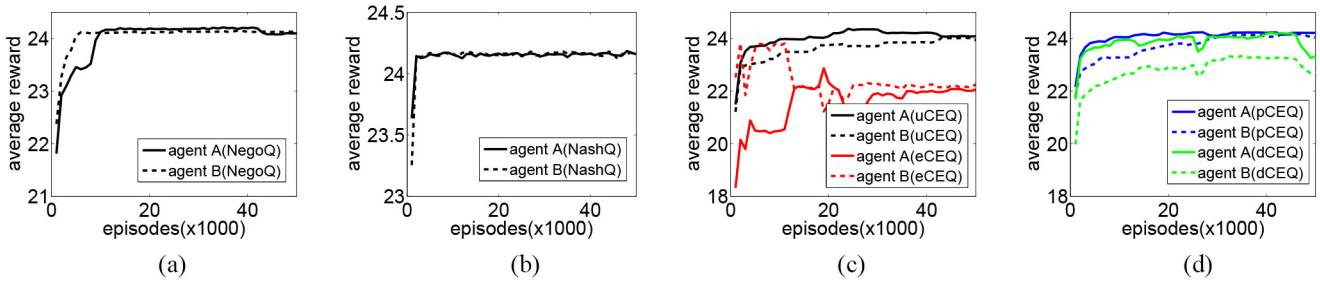


Fig. 6. Online learning performance of each algorithm in GW1. The learning rate α for NegoQ is 0.1 and $1/n(s, \vec{a})$ for NashQ and CE-Q, $\gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

V. Experiments On Grid-World Games

In this section, we compare NegoQ with the state-of-the-art equilibrium-based MARL algorithms on grid-world games, which is a commonly used test bed in MARL domain [16], [18], [27], [30]. Two equilibrium-based MARL algorithms Nash Q-learning (NashQ) [16] and correlated Q-learning (CE-Q) [18] are chosen for comparison.

We implement four versions of CE-Q according to the objective functions presented by Greenwald and Hall [18]: utilitarian CE-Q (uCE-Q), egalitarian CE-Q (eCE-Q), plutocratic CE-Q (pCE-Q), and dictatorial CE-Q (dCE-Q). The former three versions are centralized while dCE-Q is decentralized. For NashQ, we use the state-of-the-art PNS algorithm [31] for computing a Nash equilibrium in each state.

A. Experiment Settings

First, two 2-agent grid-world games are adopted to evaluate our algorithm, which are also used for experiments in related work [16], [18], [30]. Since our algorithm can be used for general n-agent ($n \geq 2$) games, we also design two 3-agent grid-world games. All of these games (GW1, GW2, GW3, and GW4) are shown in Fig. 5. In these games, agents should learn to reach their goals (possibly overlapping) within a few steps. States can be distinguished by agents' joint locations. The action set for each agent in each state is {up, down, left, right}. However, if the agents move out of the border or a collision happens, they will be placed back to their previous positions. One episode of the game is over when all agents reach their goals.

As shown in Fig. 5, the agents are denoted by A, B, and C and their goals are denoted by G_A, G_B, and G_C or an overlapping goal G. The black dashed lines in GW2 and GW4 are barriers for achieving stochastic transitions: if an agent attempts to move through one of the barriers, then with probability 0.5 this move will fail. The reward function is set as follows. Whenever an agent reaches its goal, it receives a positive reward of 100. A negative reward of -10 is received when an agent steps out of the border or a collision happens. In other cases, the reward is set to -1 in order to encourage fewer steps.

Learning algorithms are expected to acquire a winning strategy with a few steps and no collisions. In GW1, there are several pairs of pure strategy equilibrium policies, while there are exactly two in GW2 [18]. The arrow lines in Fig. 5 show one pure strategy equilibrium policy for each of the four games. We study the online learning performance of each algorithm in these four grid world games, respectively. For each algorithm, 50 experiments are repeated and each experiment is composed of 50 000 episodes. We record the average reward obtained by each algorithm in each episode. The parameters of the algorithms are set as follows.

1) Our implementation of each algorithm is on-policy and ϵ-greedy, with $\epsilon = 0.01$ and discount rate $\gamma = 0.9$.
2) The learning rate α is set to 0.1 for NegoQ.
3) In the 2-agent games, α is set to $1/n(s, \vec{a})$ for NashQ and each version of CE-Q as do in [16] and [18], where $n(s, \vec{a})$ is the number of visits to state-action pair $(s, \vec{a})$.
4) In the 3-agent games, α of NashQ and CE-Q is set to 0.1 (their performance with a variable learning rate is not satisfactory).

B. Average Rewards

Results of the four games are shown in Figs. 6–9, respectively. In these figures, the vertical axis stands for an algorithm's average reward and the horizontal axis stands for the number of episodes. Each data point of a curve is computed

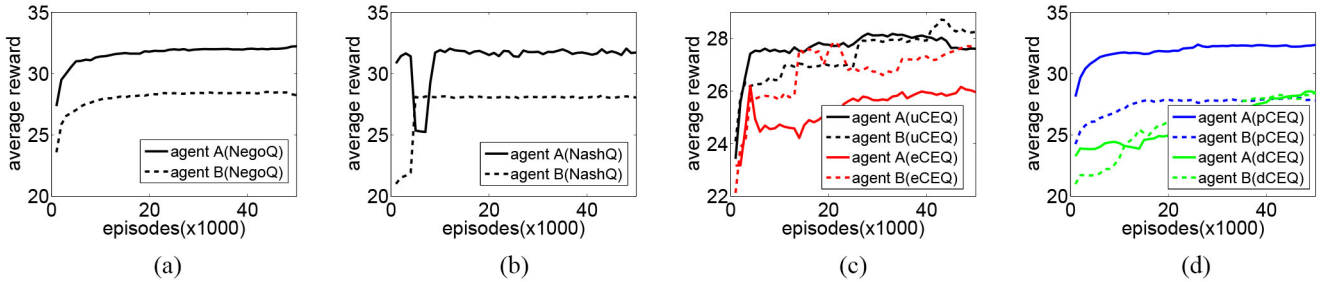


Fig. 7. Online learning performance of each algorithm in GW2. The learning rate α for NegoQ is 0.1 and $1/n(s, \vec{a})$ for NashQ and CE-Q, $\gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

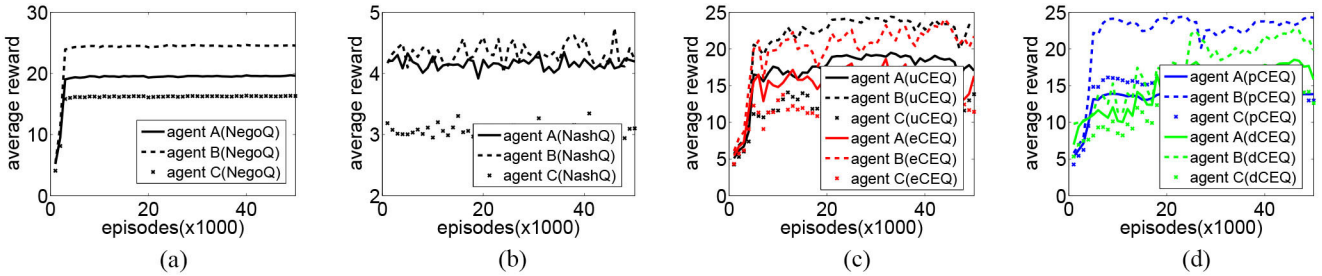


Fig. 8. Online learning performance of each algorithm in GW3. For all algorithms, $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

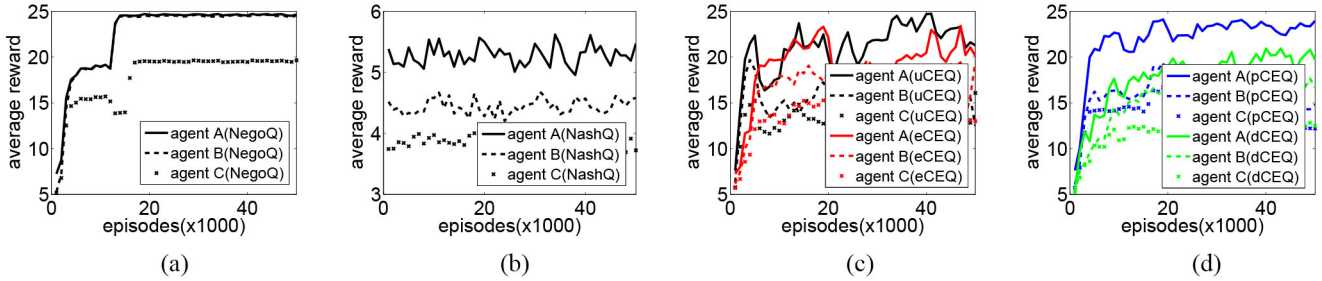


Fig. 9. Online learning performance of each algorithm in GW4. For all algorithms, $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

by averaging the values recorded in the corresponding episode of these 50 experiments. The curves of some different algorithms are plotted in the same coordinate system for saving space.

Examining Fig. 6 first, it can be found that in GW1, almost all of these algorithms finally reach a value around 24. However, eCE-Q converges to a lower value 22 for both the agents. For dCE-Q, its two curves even decrease after 45 000 episodes. In GW2, NegoQ, NashQ, and pCE-Q perform similarly, with one agent converging to 32 and the other converging to 28. Moreover, they perform more stably than the other algorithms. By carefully comparing their curves, it can be found that NashQ is less stable before converging, since a great volatility appears on agent A's learning curve in Fig. 7(b). For the other three algorithms (uCE-Q, eCE-Q, and dCE-Q), most of their curves reach relatively lower values. For example, the solid curve of eCE-Q stays below 26 almost all along the 50 000 episodes.

The advantage of NegoQ seems vague in the 2-agent games. In the 3-agent grid world games, we can find that NegoQ performs significantly better than the other algorithms. Figs. 8 and 9 show that only NegoQ plays stably in GW3 and GW4, since its learning curves are much smoother than those of the other algorithms. Furthermore, the NegoQ agents finally achieve higher values than the others. To our surprise, the performance of NashQ is not so satisfactory, with no indication of convergence observed during the 50 000 episodes.

C. Learned Policies

Above results only show the rewards obtained by each algorithm. In Markov games, it is also necessary to evaluate the policies learned by the algorithms. Thus, we record the steps taken for each agent to reach its goal in each episode. Note that for agents A and B, one pure strategy equilibrium policy will take them 4 steps in GW1, while for agents A, B, and C in the 3-agent games, one equilibrium policy will, respectively, take them 5, 4, and 6 steps in GW3 and 4, 4, and 5 steps in GW4. In GW2, an equilibrium policy will take 3 steps for one agent and 3.5 steps for the other. The equilibrium policy shown in Fig. 5(b) lets agent A go through the central grids and forces agent B to go through the barrier in the first step. If

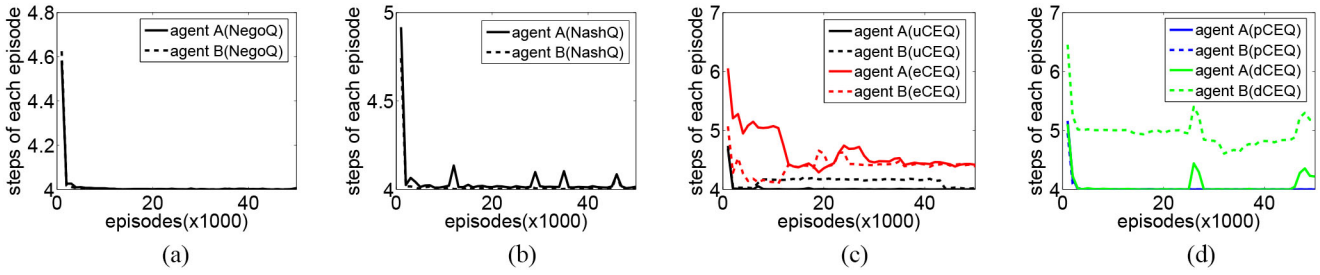


Fig. 10. Step curves of each algorithm in GW1. The learning rate α for NegoQ is 0.1 and $1/n(s, \vec{a})$ for NashQ and CE-Q, $\gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

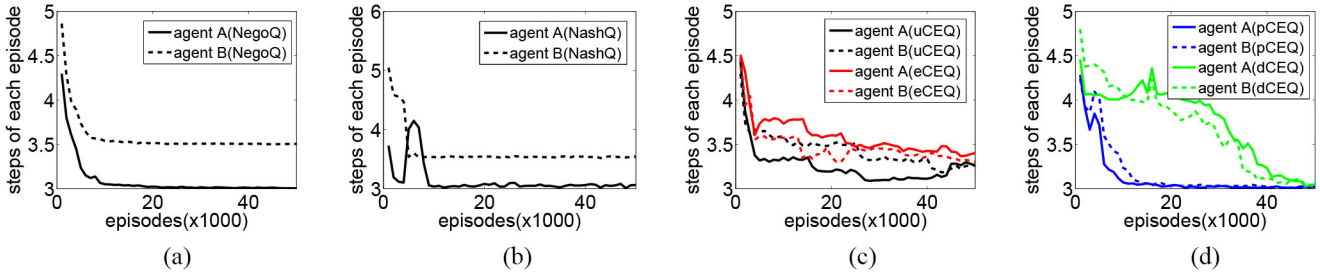


Fig. 11. Step curves of each algorithm in GW2. The learning rate α for NegoQ is 0.1 and $1/n(s, \vec{a})$ for NashQ and CE-Q, $\gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

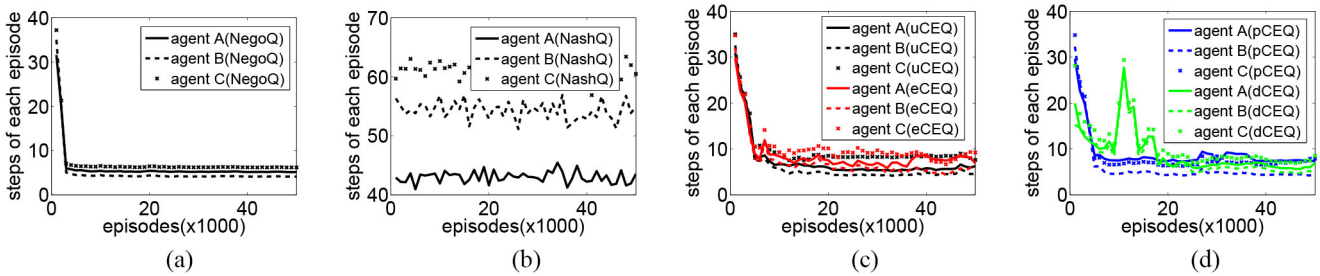


Fig. 12. Step curves of each algorithm in GW3. For all algorithms, $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

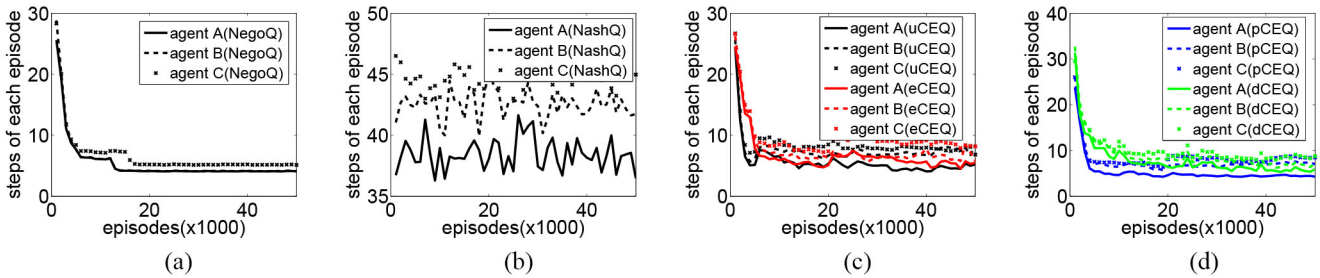


Fig. 13. Step curves of each algorithm in GW4. For all algorithms, $\alpha = 0.1, \gamma = 0.9, \epsilon = 0.01$. (a) NegoQ. (b) NashQ. (c) uCE-Q and eCE-Q. (d) pCE-Q and dCE-Q.

agent *B* is stopped by the barrier, the equilibrium policy forces it to move left in the second step. Thus, this equilibrium policy takes 3 steps for agent *A* and 3.5 steps (in expectation) for agent *B*.

Corresponding results are shown in Figs. 10–13. In Fig. 10, both the curves of NegoQ converge to a value of 4, indicating that an equilibrium policy is learned by the two NegoQ agents. NashQ, uCE-Q, and pCE-Q also converge to the optima. However, several peaks appear on the curves of NashQ and the dashed curve of uCE-Q stays above the optimal value during a long period. It seems that the other two CE-Qs fail to learn an equilibrium policy: 1) eCE-Q converges to local optimal values (about 4.5) and 2) the curves of dCE-Q rise right before the game is over (that is why its average rewards decrease at that time). In GW2, the results are similar. While NegoQ, NashQ, and pCE-Q converge close to the optimal values (3 and 3.5), the curves of the other three algorithms all reach values around 3.5.

TABLE I
TOTAL RUNTIME OF EACH ALGORITHM TO FINISH THE 50 GROUPS OF EXPERIMENTS IN GW1, GW2, GW3, AND GW4

Algorithm	GW1	GW2	GW3	GW4
NegoQ	73.70s	93.21s	999.62s	961.70s
uCE-Q	1735.78s	1530.31s	13 hours	14 hours
eCE-Q	3035.01s	2034.97s	41 hours	42 hours
pCE-Q	2332.87s	1912.96s	35 hours	38 hours
dCE-Q	1874.63s	1897.68s	41 hours	39 hours
NashQ	29 hours	16 hours	>1 weeks	>1 weeks

As shown in Figs. 12 and 13, in the two 3-agent games, NegoQ performs much better than the other algorithms, with pure strategy equilibrium policies learnt in both GW3 and GW4. In GW4, it takes more episodes for NegoQ to converge because stochastic transitions are added. The curves of the CE-Q algorithms are not so even and most of them do not converge to the optimal values. With NashQ algorithm, tens of steps are needed for the three agents to finish an episode during the whole process. As a consequence, the average rewards achieved by NashQ are much lower, which is shown in the Figs. 8 and 9.

D. Runtime Performance

We also record the total runtime of each algorithm for finishing the 50 groups of experiments in each grid world, which is shown in Table I. NegoQ runs the fastest and can complete training within 100 s in 2-agent cases and within 1000 seconds in 3-agent cases. Although the four CE-Q algorithms can complete training in one hour in GW1 and GW2, multiple hours are spent for them to finish the 3-agent games. NashQ is the most time-consuming algorithm. In GW3 and GW4, the learning process even lasts for more than one week.

The runtime of each learning algorithm is mainly determined by the corresponding equilibrium computing algorithm. In GW1 and GW2, the average time taken for the multistep negotiation process to find an equilibrium in each state is around 4μs, while in GW3 and GW4, it is around 30μs. For computing a correlated equilibrium, CE-Q has to solve one or more linear program in each state. The average time for computing a correlated equilibrium is around 120μs in GW1 and GW2, and 2 ms in GW3 and GW4. To find a Nash equilibrium, PNS algorithm [31] needs to solve a feasibility program iteratively until a Nash equilibrium is found. This feasibility program is linear in 2-agent cases and quadratic in 3-agent cases. The average time used for PNS to find a Nash equilibrium is 1ms in GW1 and GW2, and 15ms in GW3 and GW4. Therefore, a large amount of time is needed for NashQ to finish training.

E. Discussions

The above experiments show empirical convergence of NegoQ, either in environments with stochastic transitions or without. In each state, NegoQ agents may choose a PNE, an EDSP, a nonstrict EDSP, or a meta equilibrium. As a consequence, this makes the theoretical convergence proof of NegoQ difficult. However, we have proven that the former three strategy profiles all belong to the set of symmetric meta equilibria, showing the power of NegoQ from another perspective. Importantly, NegoQ provides us with a simpler and faster way to learn equilibrium policies.

The most important advantage of NegoQ is that it does not require agents to have knowledge of the others' value functions. In a general-sum Markov game, the relationship between agents is vague and thus it is better to keep them in privacy. In the real world, an agent usually is unable to observe the other agents' internal information. Another significant advantage of our algorithm is its low computational cost. Although CE-Q and NashQ have been successfully applied to 2-agent Markov games, it seems difficult for them to be applied to multiagent tasks with more than two agents due to their high computational cost. Furthermore, the average rewards achieved by CE-Q and NashQ sometimes are below the optimal values. Two reasons can cause such deviation. Firstly, the policies learnt by CE-Q and NashQ algorithms are stochastic so that it is possible that agents choose conflicting actions. Secondly, there are often multiple equilibria in a game. The existence of multiple equilibria can make it difficult for these algorithms converge to the same equilibrium policy and result in unstable learning process. In contrast, NegoQ can force all agents to choose the same strategy profile in each state.

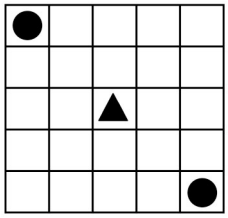


Fig. 14. Initial states of the pursuit game. Each black solid circle represents a hunter and the triangle in the central grid is the prey.

VI. EXPERIMENTS ON PURSUIT GAMES

Since the two identified strategy profiles, EDSP and nonstrict EDSP, allow for cooperation between agents (i.e., choosing strategy profiles that have higher payoffs than Nash equilibria), the second experiment is designed to investigate whether NegoQ can achieve cooperation between agents. Specifically, a widely used benchmark called pursuit game is adopted. It is a team Markov game in which there are one or more hunters and a prey. The goal of the hunters is to hunt for the prey and the goal of the prey is to escape from the hunters. If there are more than one hunter in the game, it is better for them to work cooperatively. We use the following variant of pursuit game in our experiments.

1) A prey and two hunters are placed in a 5×5 grid world.
2) The two hunters are located in the left-top and right-bottom, respectively. The prey is located in the center of the world. Fig. 14 shows their initial locations.
3) The action set of all agents in each state is {up, down, left, right, stand}. However, the prey has a 0.5 probability of failing if it moves, when it will be forced to stay in

TABLE II
AVERAGE STEPS USED FOR EACH ALGORITHM TO CATCH THE PREY IN 10 000 TESTING EPISODES OF THE PURSUIT GAME. THE FIRST LINE SHOWS SEVERAL GROUPS OF PARAMETERS IN TRAINING

Algorithm	$\gamma = 0.1, \varepsilon = 0.01$	$\gamma = 0.9, \varepsilon = 0.01$	$\gamma = 0.1, \varepsilon = 1$	$\gamma = 0.9, \varepsilon = 1$
NegoQ ($\alpha = 0.1$)	5.50 ± 2.44	4.84 ± 1.09	4.09 ± 1.51	4.20 ± 1.55
friend-Q ($\alpha = 0.1$)	5.11 ± 2.66	6.10 ± 2.94	4.16 ± 1.61	4.20 ± 1.59
JAL ($\alpha = 1/n(s, \vec{a})$)	5.79 ± 1.96	72.60 ± 75.65	4.13 ± 1.71	85.73 ± 82.82
distributed-Q	6.55 ± 2.96	7.13 ± 1.78	4.27 ± 2.07	4.38 ± 2.42

its current location. This models the situation that the hunters are faster than the prey.

4) The game is finished when any hunter catches the prey. At that time, each of the hunters will get a reward of 200.

A. Experiments and Results

Since we only care about the cooperation of the hunters, a random agent is chosen to control the behaviors of the prey. The hunters are equipped with learning algorithms. To compare with NegoQ, several MARL algorithms designed for team tasks are also tested, including JAL [22], friend Q-learning (friend-Q) [17], and distributed Q-learning (distributed-Q) [23]. All of the three algorithms are decentralized: JAL learns the Q-values of joint actions and maintains an opponent model for each of the other agents to help with selecting actions; friend-Q also learns the values of joint actions and updates Q-values based on coordination Nash equilibria; distributed-Q solves team tasks without assuming coordination and with limited computation, in which each agent only maintains the Q-value of its own actions.

Each algorithm is trained for 10 000 episodes and then tested in another 10 000 episodes. In each test episode, agents take actions according to the policy learnt during the training process with exploration factor $\epsilon = 0$. We record the steps taken for the hunters to catch the prey in each test episode and compute the average value. Several groups of parameters are chosen for each algorithm in the training process. A decreasing learning rate is needed for JAL to satisfy its convergence condition. In our implementation of JAL, the learning rate $\alpha = 1/n(s, \vec{a})$, where $n(s, \vec{a})$ is the number of visits to the state-action pair $(s, \vec{a})$. The parameters of distributed-Q only include the discount rate γ and the exploration factor ϵ. The results are listed in Table II.

In general, all algorithms can capture the prey within a few steps. However, JAL spends more than 70 steps when the discount rate γ is set to 0.9. In a further study, the performance of JAL decreases as γ increases, indicating that the value of γ has a big impact on the performance of JAL. It is difficult to tell which of the algorithms is the best because the parameters chosen have more or less impact on the results. But surprisingly, NegoQ shows its ability in team Markov games and can perform as well as these team-task-oriented algorithms.

VII. RELATED WORK

MARL has gained much attention over the last years. The wide variety of approaches proposed in MARL community integrate developments in the areas of RL, game theory, and more general direct policy search techniques [9].

In MARL, it is often difficult to maximize all agents' long-time returns at the same time. To specify a better learning goal and evaluate learning algorithms, some criteria were defined. To the best of our knowledge in the AI community, Bowling, and Veloso [30] were the first to put forth such a set of criteria (i.e., rationality and convergence). Their algorithm WoLF-IGA satisfies these criteria, but only in repeated games with two agents and two actions per agent. Subsequently, two algorithms, ReDVaLeR [32] and AWESOME [33], extended rationality and convergence to more general repeated games. The main difference between the two algorithms is that in AWESOME there is no need for agents to observe the others' mixed strategies while in ReDVaLeR such observation is required.

Later, another set of criteria called guarded optimality was proposed by Powers *et al.* [34] with the hope that algorithms would generalize well to bigger games and against more sophisticated opponents. Guarded optimality requires an learning algorithm to maximize social welfare by exploiting the agents from the target set and individually always achieve a return higher than the security value. Also, Powers *et al.* [34] proposed two algorithms, PCM(S) and PCM(A), to achieve guarded optimality. More recently, Chakraborty and Stone [35] proposed a newer set of criteria for evaluating multiagent learning algorithms in the presence of memory-bounded agents, which included convergence, targeted optimality against memory-bounded agents, and safety. Their corresponding algorithm CMLES successfully achieved the three criteria and improved previous algorithms in many aspects. For example, PCM(A) only guarantees optimality against memory-bounded agents drawn from a target set while CMLES can model any memory-bounded agent.

However, all above cited algorithms are designed for repeated games and it is difficult to use their evaluation criteria to evaluate MARL algorithms for more complicated games, such as Markov games (a.k.a. stochastic games). Currently, one of the most important approaches for learning in Markov games is equilibrium-based MARL. Littman [14] was the first to introduce Markov games into MARL and proposed the first equilibrium-based MARL algorithm minimax-Q. In minimax-Q, the state-value function is defined as the maximal joint-action-value over the values minimized by the opponent. However, it only solves two-agent zero-sum Markov games. Comparing with minimax-Q, FFQ [17] has no limit in the number of agents, in which an agents friends are assumed to work together to maximize its value, while its foes

work together to minimize its value. Based on the concept of Nash equilibrium, Hu and Wellman [15], [16] proposed Nash Q-learning (NashQ)for general-sum Markov games. The major problem of NashQ is that it has a very strict convergence condition and equilibrium calculation is computationally expensive. Since correlated equilibria are much easier to compute and allow a richer set of solutions, Greenwald and Hall [18] proposed CE-Q based on the concept of correlated equilibrium. There are also some other important algorithms for Markov games, such as optimal adaptive learning [28] and asymmetric Q-learning [27].

Rather than requiring agents to be infinitely rational, the two strategy profiles identified in this paper, EDSP and nonstrict EDSP, encourage the cooperation between agents even through they may have conflicting interest. Such cooperation has been studied well in game theory recently, such as the evolution of cooperation behaviors in networks [36], [37] and the prisoners' dilemma game [38].

VIII. Conclusion

This paper focuses on equilibrium-based MARL and its key contributions are listed as follows.

1) Three pure strategy profiles instead of mixed strategy ones have been defined as the solution concepts in equilibrium-based MARL, which are PNE, EDSP, and nonstrict EDSP. The latter two are strategy profiles from which agents can obtain higher payoffs than some pure strategy Nash equilibria. All these three identified strategy profiles haven been proven to be symmetric meta equilibria by metagame theory.
2) A multistep negotiation process has been proposed to obtain the three defined strategy profiles without sharing value functions between agents, since sharing such information is unrealistic in some domains and may raise privacy and safety issues. The key idea of this negotiation process is to allow agents to exchange their action preferences.
3) A novel MARL algorithm called NegoQ has been proposed by introducing the three identified strategy profiles and the negotiation process into MARL.
4) Through experiments, NegoQ has shown: a) empirical convergence to equilibrium policies and much higher speed than all the other algorithms (NashQ and four versions of CE-Q) in grid-world games and b) equivalent performance to team-task-oriented algorithms (JAL, friend-Q, and distributed-Q) in a team Markov game called a pursuit game, though NegoQ is not directly designed for such a specialized game.

The positive results in experiments indicate that it is possible for NegoQ to be applied in some real-world application, such as multirobot navigation and multirobot hunting. Since the joint-action space increases exponentially with the number of agents, one future direction is to address the curse of dimensionality of equilibrium-based MARL. Methods for solving large state space problems [39] may provide helpful insight. Another direction is to guarantee safety through negotiation, because some malicious opponents may intentionally not act according to the solutions established. Future work will consider identifying such opponents and establishing more robust negotiation process.

(A,B)	Confess	Deny
Confess	(-9,-9)	(0,-10)
Deny	(-10,0)	(-1,-1)

(a)

(A,B)	Confess	Deny
f1	(-9,-9)	(0,-10)
f2	(-10,0)	(-1,-1)
f3	(-9,-9)	(-1,-1)
f4	(-10,0)	(0,-10)

(b)

Fig. 15. Prisoners' dilemma and one of its metagame. The indices of agents A and B are 1 and 2, respectively. (a) Original prisoners' dilemma. (b) 1Γ of prisoners' dilemma.

Appendix A
Metagame Theory

Metagame [29] is a hypothetical game derived from the original game by assuming that one agent can know other agents' actions first and its actions are reaction functions of other agents' joint actions.

Definition A.1 (Metagame): For an n-agent ($n \geq 2$) normal-form game $\Gamma = \langle N, A_1, \ldots, A_n, U_1, \ldots, U_n \rangle$ and any $i \in N$, metagame $i\Gamma$ is a tuple $\langle N, A_1, \ldots, A_{i-1}, F_i, A_{i+1}, \ldots, A_n, U_1^{i\Gamma}, \ldots, U_n^{i\Gamma} \rangle$, where F_i is the space of agent i's reaction functions. Any function in the form of $f : A_{-i} \to A_i$ is a reaction function of agent i. For any reaction function $f \in F_i$ and any of the other agents' joint action $\vec{a_{-i}} \in A_{-i}$, $U_k^{i\Gamma}(f, \vec{a_{-i}}) = U_k(f(\vec{a_{-i}}), \vec{a_{-i}})$ for any $k \in N$.

We can see that metagame $i\Gamma$ is constructed when agent i extends its actions to reaction functions. We call Γ the basic game of $i\Gamma$. We can extend the prisoners' dilemma to its 1Γ form, which is shown in Fig. 15. In 1Γ, B keeps its action space as in the original game. For A, there are four reactive strategies for the action taken by B.

f_1: Always confess no matter which action B takes.
f_2: Always deny no matter which action B takes.
f_3: Confess when B confesses and deny when it denies.
f_4: Deny when B confesses and confess when it denies.

Recursively, by treating $i\Gamma$ as the basic game, we can construct metagames with higher orders. For example, 21Γ can be constructed from 1Γ and 321Γ can be constructed from 21Γ. For each metagame, the string before Γ is called the prefix. A special type of metagame is complete game.

Definition A.2 (Complete Game): For an n-agent normal-form game Γ, a metagame $\eta\Gamma$ is a complete game if the prefix η is a permutation of $1, \ldots, n$.

Definition A.3 (Meta Equilibrium): For a normal-form game Γ and one of the agents' joint actions $\vec{a}$, if there exists a metagame $\eta\Gamma$ and a PNE $\vec{x}$ of $\eta\Gamma$ that satisfies $\phi(\vec{x}) = \vec{a}$, $\vec{a}$ is a meta equilibrium of Γ. ϕ is the operator which maps a joint action in a metagame to its original action in the basic game.

In short, a PNE of a metagame corresponds to a meta equilibrium of the basic game. For example, in Fig. 15(b), (f_1, *Confess*) is a pure strategy equilibrium of the metagame 1Γ. Therefore, the corresponding joint action

in the basic game Γ, namely $(f_1(Confess), Confess) = (Confess, Confess)$, is a meta equilibrium of Γ.

Definition A.4 (Symmetric Meta Equilibrium): A meta equilibrium $\vec{a}$ of a normal-form game Γ is a symmetric meta equilibrium if and only if $\vec{a} \in \bigcap_{\forall \tau\Gamma} \hat{E}(\tau\Gamma)$, where $\tau\Gamma$ denotes a complete game of Γ.

We denote the set of meta equilibria of Γ by $\hat{E}(\Gamma)$. In particular, the set of meta equilibria deduced from $\eta\Gamma$ is denoted by $\hat{E}(\eta\Gamma)$. The set of symmetric meta equilibria is denoted by $Sym(\Gamma)$. Here, we give a sufficient and necessary condition of meta equilibrium [40], which provides us with an effective way to compute meta equilibria.

Definition A.5 (A Sufficient and Necessary Condition of Meta Equilibrium): For an n-agent normal-form game Γ, a joint action $\vec{a}$ is called a meta equilibrium from a metagame $k_1k_2\cdots k_r\Gamma$ if and only if for any i there holds

$$U_i(\vec{a}) \geq \min_{\vec{a_{P_i}}} \max_{a_i} \min_{\vec{a_{S_i}}} U_i(\vec{a_{P_i}}, a_i, \vec{a_{S_i}}, \vec{a_{N_i}}) \tag{14}$$

where P_i is the set of agents listed before sign i in the prefix $k_1k_2\cdots k_r$, S_i is the set of agents listed after sign i and N_i is the set of agents which do not appear in the prefix. If i is not listed in the prefix, let P_i be $\emptyset$ and S_i be the set of all agents listed in the prefix. Therefore, $\vec{a_{P_i}}$, $\vec{a_{S_i}}$, $\vec{a_{N_i}}$ represent the joint actions of agents in set P_i, S_i and N_i, respectively.

Thus, to compute a meta equilibrium, there is no need to extend a normal-form game to its meta game and find the corresponding Nash equilibria of that meta game. For example, if Γ is a 3-agent normal-form game, for computing the set $\hat{E}(321\Gamma)$, we need to find the joint action $\vec{a} = (a_1, a_2, a_3)$ which satisfies

$$\begin{cases} U_1(a_1,a_2,a_3) \geq \min_{a'_3}\min_{a'_2}\max_{a'_1} U_1(a'_1,a'_2,a'_3), \\ U_2(a_1,a_2,a_3) \geq \min_{a'_3}\max_{a'_2}\min_{a'_1} U_2(a'_1,a'_2,a'_3), \\ U_3(a_1,a_2,a_3) \geq \max_{a'_3}\min_{a'_2}\min_{a'_1} U_3(a'_1,a'_2,a'_3). \end{cases}$$

Recall that one issue of the three strategy profiles defined in this paper is that they may not exist in a game. We choose a meta equilibrium as an alternative solution concept in such situation. The reasons are as follows. Firstly, a meta equilibrium is a pure strategy equilibrium and always exists. Secondly, as shown in (14), the utility of a meta equilibrium for each agent is always higher than a threshold value. Lastly, a meta equilibrium is consistent with the three identified strategy profiles since they also belong to the category of meta equilibria. Appendix B shows that meta equilibria deduced from complete games always exist. Therefore, we can first choose a complete game and then find a meta equilibrium through negotiation according to (14). Corresponding algorithm is given in Algorithm 5.

Algorithm 5: Negotiation for a Meta Equilibrium Set

Input: A normal-form game $\langle N, \{A_i\}_{i=1}^n, \{U_i\}_{i=1}^n \rangle$

/* But agent i only knows N, $\{A_i\}_{i=1}^n$, and U_i. */

The set of meta equilibria $J_{meta} = \emptyset$;

Determine a complete game randomly or by any other possible mechanism;

$\eta \leftarrow$ the prefix of the complete game chosen;

$P_i \leftarrow$ the set of indices listed before 'i' in η;

$S_i \leftarrow$ the set of indices listed after 'i' in η;

$threValue \leftarrow \min_{\vec{a_{P_i}}} \max_{a_i} \min_{\vec{a_{S_i}}} U_i(\vec{a_{P_i}}, a_i, \vec{a_{S_i}})$;

foreach *joint action* $\vec{a} \in A$ **do**

  if $U_i(\vec{a}) \geq threValue$ **then**

    $J_{meta} \leftarrow J_{meta} \cup \{\vec{a}\}$;

/* QUESTIONING THREAD: */

foreach *joint action* $\vec{a} \in J_{meta}$ **do**

  Ask all the other agents whether $\vec{a}$ is also in their J_{meta} sets;

  if *one of the answers is 'no'* **then**

    $J_{meta} \leftarrow J_{meta} \setminus \{\vec{a}\}$;

/* ANSWERING THREAD: */

foreach *joint action* $\vec{a}'$ *received from the other agents* **do**

  if $\vec{a}'$ *is in* J_{meta} **then**

    Send 'yes' back to the agent;

  else

    Send 'no' back to the agent;

APPENDIX B
MATHEMATICAL PROOFS

In this section, we prove the theorems given in this paper. We first give three lemmas to show the formal relationship between a basic game and its metagames. They are also used for proving the theorems in our paper. Denote an element of a normal-form game Γ by $\vec{g}_{\vec{a}} = (U_1(\vec{a}), U_2(\vec{a}), \ldots, U_n(\vec{a}))$, where $\vec{a}$ is the joint action corresponds to this element. For any agent $i \in N$ and any action $a_i \in A_i$, define a set $G^i_\Gamma(a_i) = \{\vec{g}_{(a_i, \vec{a^j_{-i}})} \mid \vec{a^j_{-i}} \in A_{-i}, j = 1, 2, \ldots, |A_{-i}|\}$. It can be found that $G^i_\Gamma(a_i)$ contains all game elements in the $(n-1)$-dimensional matrix corresponding to the action a_i of agent i. We call this $(n-1)$-dimensional matrix a "row" of agent i. From the viewpoint of agent i, $\Gamma = \bigcup_{a \in A_i} G^i_\Gamma(a)$. For a joint action $\vec{a} \in A$, denote agent i's action in $\vec{a}$ by $\vec{a}(i)$. The following lemma indicates that any reaction function $f \in F_i$ of agent i corresponds to one of its rows in $i\Gamma$ and vice versa.

Lemma B.1: Let Γ be an n-agent normal-form game, for any agent $i \in N$, metagame $i\Gamma$ can be represented by a $|A_1| \times \cdots \times |A_{i-1}| \times |A_i|^{|A_{-i}|} \times |A_{i+1}| \times \cdots \times |A_n|$ game matrix, where each row for agent i is in the form of $(\vec{g}_{(a_i^1, \vec{a_{-i}^1})}, \vec{g}_{(a_i^2, \vec{a_{-i}^2})}, \ldots, \vec{g}_{(a_i^m, \vec{a_{-i}^m})})$, where $a_i^j \in A_i$, $\vec{a^j_{-i}} \in A_{-i}$ for all $j = 1, \ldots, m$ and $m = |A_{-i}|$.

Proof: Let $\mathscr{F} = \{f | f : A_{-i} \to A_i\}$ and $\mathscr{R} = \{(\vec{g}_{(a_i^1, \vec{a_{-i}^1})}, \vec{g}_{(a_i^2, \vec{a_{-i}^2})}, \ldots, \vec{g}_{(a_i^m, \vec{a_{-i}^m})}) | (a^1, \ldots, a^m) \in \underbrace{A_i \times \cdots \times A_i}_{m}\}$. Thus,

$\mathscr{F}$ is the set of all reaction functions of agent i and we want to show that $\mathscr{R}$ is the set of all rows in metagame $i\Gamma$.

Let $\varphi : \mathscr{F} \to \mathscr{R}, f \mapsto (\overrightarrow{g}_{(a_i^1, \overrightarrow{a_{-i}^1})}, \overrightarrow{g}_{(a_i^2, \overrightarrow{a_{-i}^2})}, \ldots, \overrightarrow{g}_{(a_i^m, \overrightarrow{a_{-i}^m})})$. On one hand, for any $(\overrightarrow{g}_{(a_i^1, \overrightarrow{a_{-i}^1})}, \overrightarrow{g}_{(a_i^2, \overrightarrow{a_{-i}^2})}, \ldots, \overrightarrow{g}_{(a_i^m, \overrightarrow{a_{-i}^m})}) \in \mathscr{R}$, we can construct a function $f{:}A_{-i} \to A_i, a_i^j \mapsto \overrightarrow{a_{-i}^j}$. Obviously, this is a reaction function, namely $f \in \mathscr{F}$. Therefore, φ is a surjection.

On the other hand, $\varphi(f)$ and $\varphi(h)$ must be distinct if $f, h \in \mathscr{F}$ are distinct. Otherwise, assume that $f \neq h$ and $\varphi(f) = \varphi(h)$, we will conclude $(\overrightarrow{g}_{(f(\overrightarrow{a_{-i}^1}), \overrightarrow{a_{-i}^1})}, \overrightarrow{g}_{(f(\overrightarrow{a_{-i}^2}), \overrightarrow{a_{-i}^2})}, \ldots, \overrightarrow{g}_{(f(\overrightarrow{a_{-i}^m}), \overrightarrow{a_{-i}^m})}) = (\overrightarrow{g}_{(h(\overrightarrow{a_{-i}^1}), \overrightarrow{a_{-i}^1})}, \overrightarrow{g}_{(h(\overrightarrow{a_{-i}^2}), \overrightarrow{a_{-i}^2})}, \ldots, \overrightarrow{g}_{(h(\overrightarrow{a_{-i}^m}), \overrightarrow{a_{-i}^m})})$. Therefore, $f(\overrightarrow{a_{-i}}) = h(\overrightarrow{a_{-i}})$ for any joint action $\overrightarrow{a_{-i}} \in A_{-i}$. This is a contradiction with the assumption that $f \neq h$. Thus, φ is also an injection.

Therefore, φ is a bijection between $\mathscr{F}$ and $\mathscr{R}$. The total number of distinct reaction functions is $|A_i|^{|A_{-i}|}$. Thus, the number of agent i's rows in $i\Gamma$ is $|A_i|^{|A_{-i}|}$, and $\mathscr{R}$ is the set of all rows of agent i in $i\Gamma$. ■

Lemma B.2: Let Γ be an n-agent ($n \geq 2$) normal-form game, for an agent $i \in N$, $G_{i\Gamma}^j(a_j) = G_{\Gamma}^j(a_j)$ holds for any agent $j \neq i$ and for any action $a_j \in A_j$.

This lemma is a direct corollary of Lemma B.1 and shows that when constructing metagame $i\Gamma$ from Γ, each of the other agents' action space remains unchanged.

Lemma B.3: For an n-agent ($n \geq 2$) normal-form game Γ, denote the set of pure strategy Nash equilibria by $E(\Gamma)$, then $E(\Gamma) \subseteq \hat{E}(i\Gamma)$ for any $i \in N$.

Proof: The idea of proving this lemma is to map a PNE to a meta equilibrium derived from $i\Gamma$. Suppose $\overrightarrow{e} \in E(\Gamma)$ is a PNE of Γ. For any $i \in N$, we can construct a reaction function $f_i{:}A_{-i} \to A_i$ which satisfies $f_i(\overrightarrow{a_{-i}}) = \overrightarrow{e}\ (i)$ for any $\overrightarrow{a_{-i}} \in A_{-i}$. By Lemma B.1, f_i corresponds to a row of agent i in $i\Gamma$.

Then consider the joint action $\overrightarrow{x} = (\overrightarrow{e}\ (1), \ldots, f_i, \ldots, \overrightarrow{e}\ (n))$ in $i\Gamma$. We can find that for any $j \neq i$, there holds

$$\begin{aligned} U_j^{i\Gamma}(\overrightarrow{x}) &= U_j^{i\Gamma}(\overrightarrow{x}\ (1), \ldots, \overrightarrow{x}\ (n)) \\ &= U_j^{i\Gamma}(\overrightarrow{e}\ (1), \ldots, f_i, \ldots, \overrightarrow{e}\ (n)) \\ &= U_j(\overrightarrow{e}\ (1), \ldots, \overrightarrow{e}\ (i), \ldots, \overrightarrow{e}\ (n)) \\ &= \max_{a_j \in A_j} U_j(\overrightarrow{e}\ (1), \ldots, a^j, \ldots, \overrightarrow{e}\ (n)) \\ &= \max_{a_j \in A_j} U_j^{i\Gamma}(\overrightarrow{x}\ (1), \ldots, a^j, \ldots, \overrightarrow{x}\ (n)). \end{aligned}$$

For agent i, since the domain of agent i's reaction function is A_i, we have

$$\begin{aligned} U_i^{i\Gamma}(\overrightarrow{x}) &= U_i^{i\Gamma}(\overrightarrow{x}\ (1), \ldots, \overrightarrow{x}\ (n)) \\ &= U_i^{i\Gamma}(\overrightarrow{e}\ (1), \ldots, f_i, \ldots, \overrightarrow{e}\ (n)) \\ &= U_i(\overrightarrow{e}\ (1), \ldots, \overrightarrow{e}\ (i), \ldots, \overrightarrow{e}\ (n)) \\ &= \max_{a_i \in A_i} U_i(\overrightarrow{e}\ (1), \ldots, a^i, \ldots, \overrightarrow{e}\ (n)) \\ &= \max_{f \in F_i} U_i^{i\Gamma}(\overrightarrow{e}\ (1), \ldots, f, \ldots, \overrightarrow{e}\ (n)). \end{aligned}$$

Therefore, $\overrightarrow{x}$ is a PNE of $i\Gamma$. Correspondingly, $\overrightarrow{e}$ is also a meta equilibrium which can be derived from $i\Gamma$. ■

Theorem B.1: Any PNE in a normal-form game is also a symmetric meta equilibrium.

Proof: Suppose Γ is an n-agent normal-form game. By Lemma B.3, we have $E(\Gamma) \subseteq \hat{E}(1\Gamma)$. Since $\hat{E}(1\Gamma)$ also represents the set of pure strategy Nash equilibria of 1Γ, we can also get $\hat{E}(1\Gamma) \subseteq \hat{E}(21\Gamma)$ by applying Lemma B.3 to 1Γ. Therefore, we have

$$E(\Gamma) \subseteq \hat{E}(1\Gamma) \subseteq \hat{E}(21\Gamma) \subseteq \cdots \subseteq \hat{E}(n(n-1)\cdots 21\Gamma).$$

Analogously, we can prove $E(\Gamma)$ is a subset of all the other complete games.

Note that $Sym(\Gamma) = \bigcap_{\forall \tau\Gamma} \hat{E}(\tau\Gamma)$, where $\tau\Gamma$ denotes a complete game. Therefore, for any PNE $\overrightarrow{e} \in E(\Gamma)$, we have $\overrightarrow{e} \in Sym(\Gamma)$. ■

Theorem B.2: Any nonstrict EDSP in a normal-form game is a symmetric meta equilibrium.

Proof: We prove this theorem by finding reaction functions which map a nonstrict EDSP to a symmetric meta equilibrium. Suppose $\overrightarrow{x}$ is a nonstrict EDSP in an n-agent normal-form game Γ. According to the definition of nonstrict EDSP, there must exist a PNE $\overrightarrow{e_i}$ for any $i \in N$, which satisfies $U_i(\overrightarrow{x}) \geq U_i(\overrightarrow{e_i})$. We need to prove that in each complete game of Γ, there exist an NE that corresponds to $\overrightarrow{x}$. We first consider the metagame $n(n-1)\cdots 21\Gamma$.

Constructing 1Γ based on Γ, we can find a reaction function f_1 of agent 1 in 1Γ

$$f_1(\overrightarrow{a_{-1}}) = \begin{cases} \overrightarrow{x}\ (1) & \text{if } \overrightarrow{a_{-1}} = (\overrightarrow{x}\ (2), \ldots, \overrightarrow{x}\ (n)) \\ \overrightarrow{e_j}\ (1) & \text{else if } \overrightarrow{a_{-1}} = (\overrightarrow{e_j}\ (2), \ldots, \overrightarrow{e_j}\ (n))(j > 1) \\ \overrightarrow{e_1}\ (1) & \text{otherwise.} \end{cases}$$

Then we extend 1Γ to 21Γ and we can find a reaction function f_2 of agent 2

$$f_2(\overrightarrow{a_{-2}}) = \begin{cases} \overrightarrow{x}\ (2) & \text{if } \overrightarrow{a_{-2}} = (f_1, \overrightarrow{x}\ (3), \ldots, \overrightarrow{x}\ (n)) \\ \overrightarrow{e_j}\ (2) & \text{else if } \overrightarrow{a_{-2}} = (f_1, \overrightarrow{e_j}\ (3), \ldots, \overrightarrow{e_j}\ (n)) \\ & \quad (j > 2) \\ \overrightarrow{e_1}\ (2) & \text{else if } \overrightarrow{a_{-2}} \text{ is in the form of } (f_1, \ldots) \\ \overrightarrow{e_2}\ (2) & \text{otherwise.} \end{cases}$$

Iteratively, when we construct the metagame $i(i-1)\cdots 21\Gamma (2 < i \leq N)$, we can define a reaction function of agent i

$$f_i(\overrightarrow{a_{-i}}) = \begin{cases} \overrightarrow{x}\ (i) & \text{if } \overrightarrow{a_{-i}} = (f_1, \ldots, f_i, \ldots, \overrightarrow{x}\ (i+1), \ldots, \\ & \quad \overrightarrow{x}\ (n)) \\ \overrightarrow{e_j}\ (i) & \text{else if } \overrightarrow{a_{-i}} = (f_1, \ldots, f_{i-1}, \overrightarrow{e_j}\ (i+1), \ldots, \\ & \quad \overrightarrow{e_j}\ (n))(j > i) \\ \overrightarrow{e_k}\ (i) & \text{else if } \overrightarrow{a_{-i}} \text{ is in the form of} \\ & \quad (f_1, f_2, \ldots, f_{k-1}, \ldots)(1 \leq k < i) \\ \overrightarrow{e_i}\ (i) & \text{otherwise.} \end{cases}$$

Now consider the action $(f_1, f_2, \ldots, f_n)$ in metagame $n(n-1)\cdots 21\Gamma$, for any agent $i \in N$, obviously we have

$$\begin{aligned} &U_i^{n(n-1)\cdots 1\Gamma}(f_1, f_2, \ldots, f_n) \\ &= U_i^{(n-1)\cdots 1\Gamma}(f_1, f_2, \ldots, \vec{x}\ (n)) \\ &= U_i^{(n-2)\cdots 1\Gamma}(f_1, f_2, \ldots, \vec{x}\ (n-1), \vec{x}\ (n)) \\ &= \cdots = U_i(\vec{x}). \end{aligned}$$

For any other joint action $(f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \ldots, f_n)$ that satisfies $f_i' \neq f_i$, we can get

$$\begin{aligned} &U_i^{n(n-1)\cdots 1\Gamma}(f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \ldots, f_n) \\ &= U_i^{(n-1)\cdots 1\Gamma}(f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \ldots, f_{n-1}, \vec{e_i}\ (n)) \\ &= U_i^{(n-2)\cdots 1\Gamma}(f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \ldots, f_{n-2}, \vec{e_i}\ (n-1), \vec{e_i}\ (n)) \\ &= \cdots \\ &= U_i(\vec{e_i}\ (1), \ldots, a_i', \ldots, \vec{e_i}\ (n)). \end{aligned}$$

Here, $a_i' \in A_i$ is the original action of f_i' in the basic game Γ. Obviously

$$\begin{aligned} &U_i(\vec{e_i}\ (1), \ldots, a_i', \ldots, \vec{e_i}\ (n)) \\ &\quad \leq U_i(\vec{e_i}\ (1), \ldots, \vec{e_i}\ (i), \ldots, \vec{e_i}\ (n)) \leq U_i(\vec{x}). \end{aligned}$$

Note that this is true for any $i \in N$. Therefore, $(f_1, f_2, \ldots, f_n)$ is a PNE of metagame $n(n-1)\cdots 21\Gamma$. Correspondingly, $\vec{x}$ is a meta equilibrium which can be derived from $n(n-1)\cdots 21\Gamma$. For each of the other complete games, the proof is similar and we do not repeat. ■

Theorem B.3 (Existence of Meta Equilibrium): For any normal-form game defined in Definition 2, it has at least one meta equilibrium.

Proof: We prove this theorem by finding a metagame in which there always exists at least one PNE. Suppose Γ is an n-agent normal-form game. First, we extend Γ to 1Γ. We can construct a reaction function $f_1{:}A_{-1} \rightarrow A_1$ which satisfies $\forall\ \vec{a_{-1}} \in A_{-1}, f_1(\vec{a_{-1}}) = \arg\max_{a_1} U_1(a_1, \vec{a_{-1}})$. By Lemma B.1, f_1 corresponds to one row of agent 1 in 1Γ in which agent 1's action is a best response to each of the other agents' joint actions.

Treating 1Γ as the basic game, we can extend it to 21Γ. Similarly, we can find a row of agent 2 in 21Γ, where the action of agent 2 is the best response to all of the other agents' joint actions. Obviously, in this row, we can find an $(n-2)$-dimensional matrix where the action of agent 1 is f_1, since its action space remains unchanged from 1Γ to 21Γ. Therefore, in this $(n-2)$-dimensional matrix, both the actions of agent 1 and agent 2 are a best response to all of the other $(n-1)$ agents' joint actions.

Iteratively, by constructing metagames step by step, we can finally find a 0-dimensional matrix (namely a joint action) in $n(n-1)\cdots 21\Gamma$, where each agent's action is a best response to the other agents' joint action. Therefore, there must be a PNE in the metagame $n(n-1)\cdots 21\Gamma$. ■

Note that $n(n-1)\cdots 21\Gamma$ is a complete game. In fact, we can prove that there always exists a Nash equilibrium in any complete game in a similar way.

REFERENCES

[1] S. Liemhetcharat and M. Veloso, "Synergy graphs for configuring robot team members," in *Proc. 12th Int. Conf. Autonom. Agents Multi-Agent Syst.*, 2013, pp. 111–118.

[2] S. Pedersen, B. Foss, I. Schjølberg, and J. Tjønnås, "MAS for manufacturing control: A layered case study," in *Proc. 11th Int. Conf. Autonom. Agents Multiagent Syst.*, 2012, pp. 1169–1170.

[3] F. Bernardo, R. Agusti, J. Perez-Romero, and O. Sallent, "Intercell interference management in OFDMA networks: A decentralized approach based on reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 6, pp. 968–976, Nov. 2011.

[4] Y. Zheng, K. Xiang, D. Li, and A. V. Vasilakos, "Directional routing and scheduling for green vehicular delay tolerant networks," *Wireless Netw.*, vol. 19, no. 2, pp. 161–173, 2013.

[5] M. A. Khan, H. Tembine, and A. V. Vasilakos, "Game dynamics and cost of learning in heterogeneous 4G networks," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 1, pp. 198–213, Jan. 2012.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[7] A. F. Atlasis, N. H. Loukas, and A. V. Vasilakos, "The use of learning algorithms in ATM networks call admission control problem: A methodology," *Comput. Netw.*, vol. 34, no. 3, pp. 341–353, 2000.

[8] A. V. Vasilakosa, M. P. Saltouros, A. F. Atlassis, and W. Pedrycz, "Optimizing QoS routing in hierarchical ATM networks using computational intelligence techniques," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 33, no. 3, pp. 297–312, Aug. 2003.

[9] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[10] A. V. Vasilakos and G. I. Papadimitriou, "A new approach to the design of reinforcement schemes for learning automata: Stochastic estimator learning algorithm," *Neurocomputing*, vol. 7, no. 3, pp. 275–297, 1995.

[11] P. Vrancx, K. Verbeeck, and A. Nowé, "Decentralized learning in Markov games," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 976–981, Aug. 2008.

[12] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, "Generalized learning automata for multi-agent reinforcement learning," *AI Commun.*, vol. 23, no. 4, pp. 311–324, 2010.

[13] L. MacDermed, K. S. Narayan, and L. Weiss, "Quick polytope approximation of all correlated equilibria in stochastic games," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, pp. 707–712.

[14] M. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. 11th Int. Conf. Mach. Learn.*, 1994, pp. 157–163.

[15] J. Hu and M. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proc. 15th Int. Conf. Mach. Learn.*, 1998, pp. 242–250.

[16] J. Hu and M. Wellman, "Nash Q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, Nov. 2003.

[17] M. Littman, "Friend-or-foe Q-learning in general-sum games," in *Proc. 18th Int. Conf. Mach. Learn.*, 2001, pp. 322–328.

[18] A. Greenwald, K. Hall, and R. Serrano, "Correlated Q-learning," in *Proc. 20th Int. Conf. Mach. Learn.*, Washington, DC, USA, 2003, pp. 242–249.

[19] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in grid computing," *Future Gener. Comput. Syst.*, vol. 27, no. 5, pp. 430–439, 2011.

[20] B. An, V. Lesser, D. Westbrook, and M. Zink, "Agent-mediated multi-step optimization for resource allocation in distributed sensor networks," in *Proc. 10th Int. Conf. Autonom. Agents Multiagent Syst.*, 2011, pp. 609–616.

[21] B. An, F. Douglis, and F. Ye, "Heuristics for negotiation schedules in multi-plan optimization," in *Proc. 7th Int. Joint Conf. Autonom. Agents Multiagent Syst.*, 2008, pp. 551–558.

[22] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proc. Nat. Conf. Artif. Intell.*, 1998, pp. 746–752.

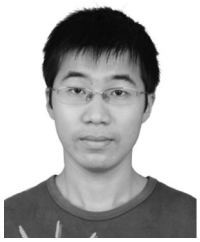
[23] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 535–542.

[24] T. Sandholm, "Perspectives on multiagent learning," *Artif. Intell.*, vol. 171, no. 7, pp. 382–391, 2007.

[25] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Dept. Comput. Sci., King's College, Cambridge, U.K., 1989.

[26] R. J. Aumann, "Subjectivity and correlation in randomized strategies," *J. Math. Econ.*, vol. 1, no. 1, pp. 67–96, 1974.

[27] V. Kononen, "Asymmetric multiagent reinforcement learning," in *Proc. IEEE/WIC Int. Conf. Intell. Agent Technol.*, 2003, pp. 336–342.
[28] X. Wang and T. Sandholm, "Reinforcement learning to play an optimal nash equilibrium in team Markov games," *Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 15, pp. 1571–1578, Vancourver, Canada, 9–14 Dec. 2002.
[29] N. Howard, *Paradoxes of Rationality: Games, Metagames, and Political Behavior*. Cambridge Massachusetts, USA: MIT Press, 1971.
[30] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artif. Intell.*, vol. 136, no. 2, pp. 215–250, 2002.
[31] R. Porter, E. Nudelman, and Y. Shoham, "Simple search methods for finding a nash equilibrium," *Games Econ. Behav.*, vol. 63, no. 2, pp. 642–662, 2008.
[32] B. Banerjee and J. Peng, "Performance bounded reinforcement learning in strategic interactions," in *Proc. 19th Nat. Conf. Artif. Intell.*, vol. 4. 2004, pp. 2–7.
[33] V. Conitzer and T. Sandholm, "AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents," *Mach. Learn.*, vol. 67, nos. 1–2, pp. 23–43, 2007.
[34] R. Powers, Y. Shoham, and T. Vu, "A general criterion and an algorithmic framework for learning in multi-agent systems," *Mach. Learn.*, vol. 67, nos. 1–2, pp. 45–76, 2007.
[35] D. Chakraborty and P. Stone, "Multiagent learning in the presence of memory-bounded agents," *Autonom. Agents Multi-Agent Syst.*, vol. 28, no. 2, pp. 182–213, 2014.
[36] J. Gómez-Gardenes, I. Reinares, A. Arenas, and L. M. Floría, "Evolution of cooperation in multiplex networks," *Sci. Rep.*, vol. 2, p. 620, Aug. 2012.
[37] Z. Wang, C.-Y. Xia, S. Meloni, C.-S. Zhou, and Y. Moreno, "Impact of social punishment on cooperative behavior in complex networks," *Sci. Rep.*, vol. 3, p. 3055, Oct. 2013.
[38] M. Perc and Z. Wang, "Heterogeneous aspirations promote cooperation in the prisoner's dilemma game," *PLoS ONE*, vol. 5, no. 12, p. e15117, 2010.
[39] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, "Learning multi-agent state space representations," in *Proc. 9th Int. Conf. Autonom. Agents Multiagent Syst.*, Toronto, ON, Canada, 2010, pp. 715–722.
[40] L. Thomas, *Games, Theory, and Applications*. Chichester, U.K.: E. Horwood, 1984.



Yujing Hu received the B.S. degree in computer science and technology from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2010, where he is currently pursuing the Ph.D. degree in computer application technology.

His current research interests include reinforcement learning, multiagent learning, and game theory.

Yang Gao (M'05) received the Ph.D. degree in computer software and theory from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2000.

He is a Professor with the Department of Computer Science and Technology, Nanjing University. His current research interests include artificial intelligence and machine learning. He has published over 50 papers in top conferences and journals in and out of China.

Bo An (M'09) received the Ph.D degree in computer science from the University of Massachusetts, Amherst, MA, USA, in 2011.

He is currently an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests include artificial intelligence, multiagent systems, game theory, automated negotiation, resource allocation, and optimization. He has published over 30 refereed papers at top conferences, including AAMAS, IJCAI, and AAAI and journals such as IEEE Transactions.

Dr. An was the recipient of the 2010 IFAAMAS Victor Lesser Distinguished Dissertation Award, an Operational Excellence Award from the Commander, First Coast Guard District of the United States, the Best Innovative Application Paper Award at AAMAS'2012, and the 2012 INFORMS Daniel H. Wagner Prize for Excellence in Operations Research Practice. He is a member of the Board of directors of IFAAMAS.