Clique-based Cooperative Multiagent Reinforcement Learning Using Factor Graphs

Zhen Zhang Dongbin Zhao

***Abstract*—In this paper, we propose a clique-based sparse reinforcement learning (RL) algorithm for solving cooperative tasks. The aim is to accelerate the learning speed of the original sparse RL algorithm and to make it applicable for tasks decomposed in a more general manner. First, a transition function is estimated and used to update the Q-value function, which greatly reduces the learning time. Second, it is more reasonable to divide agents into cliques, each of which is only responsible for a specific subtask. In this way, the global Q-value function is decomposed into the sum of several simpler local Q-value functions. Such decomposition is expressed by a factor graph and exploited by the general max-plus algorithm to obtain the greedy joint action. Experimental results show that the proposed approach outperforms others with better performance.**

***Index Terms*—Multiagent reinforcement learning, factor graph, max-plus algorithm, clique-based decomposition.**

I. Introduction

THE concept of agent plays an important role in the design of artificial intelligence. An agent can be viewed as a computational entity that can perceive its environment through sensors, make decisions according to a prior knowledge and sensed information, and act upon its environment through actuators[1]. The design goal of agent is to optimize some performance index. For example, a traffic signal light agent determines its phase and cycle time according to the traffic information transmitted from the sensors of nearby lanes[2], with the aim of minimizing the average waiting time or queues of the vehicles crossing the intersection[3] or entering the freeway ramps[4]. In many cases, an agent is not standalone but connected with others, and agents interact with each other to affect the environment together. Sometimes, each agent can only acquire the states of its nearby environment and the behaviors of its neighbor agents. Such a system is called multiagent system (MAS)[5]. If agents have common interest and coordinate to fulfill a task, they are cooperative ones, e.g., wireless network agents cooperate to formulate a stable grand coalition formation to yield significant gains with respect to average rates per link[6]. Otherwise, if each agent only pursues the interest of its own, they are competitive ones, e.g., the agents in one-on-one combat games compete with each other to make the maximum profit of each individual[7−8]. In this paper, we only deal with cooperative agents. In MAS, an agent has to learn its strategy from the environment and the others. This makes the learning system more unstable to converge. Thus, an important problem in MAS is how to make agents achieve cooperation with partial state information and the behaviors of their neighbors[9−10].

The strategies of the agents can be programmed in advance, if there is sufficient a priori knowledge about the problem. However, in many cases, the environment may change over time, which makes the hardwired strategies hardly flexible. Thus it is necessary for agents to learn strategies on their own.

Reinforcement learning (RL)[11] is a natural way to design adaptive agents. Unlike supervised learning, RL learns by trial-and-error without an explicit teacher. This is very important for designing agent, especially in case where a priori knowledge is limited. A strategy learned by RL is represented by a value function or a Q-value function. A value function maps each state to some value as an estimation of the state′s goodness in the long run, while a Q-value function maps each state-action pair to some value as an estimation of the action′s goodness in the long run. When RL is employed, a transition function can be estimated and used to update the value function online. Previous studies have shown that this can reduce the learning time greatly[12].

The learning dynamics or convergence of multiagent reinforcement learning (MARL) is the theoretical foundation. The early analysis on the dynamics of RL in cooperative MAS was done by Claus and Boutilier[13]. They analyzed the dynamics of independent learners in a two-agent repeated game. Besides, Tuyls[14] analyzed independent Q-learning (IQL) through evolutionary game theory. Gomes and Kowalczyk[15] analyzed IQL with ε-greedy exploration. Kianercy and Galstyan[16] analyzed IQL with Boltzmann exploration. These results have given inspiration for designing MARL algorithms. In fact, most theoretical results on MARL are limited to repeated games. Drawn from Markov decision process and game theory, stochastic games are proposed as a general framework for studying MARL[17−18]. Under this framework, many MARL algorithms have been presented. Examples are minimax-Q[17], Friend-or-foe[19], Nash-Q[20], IGA[21], and Wolf-PHC[22]. As the number of agents increases, the state space and the joint action space grow rapidly, which is the so-called problem of curse of dimensionality in MARL. Researchers have managed to suggest many algorithms and studied them in different domains. Adaptive dynamic programming (ADP)[23−29] is an effective approach to relieve such problem by using neural networks to approximate the value function and the policy. ADP has been applied to many fields such as air-fuel ratio

Manuscript received September 26, 2013; accepted February 24, 2014. This work was supported by National Natural Science Foundation of China (61273136, 61034002), Beijing Natural Science Foundation (4122083), and Visiting Professorship of Chinese Academy of Sciences. Recommended by Associate Editor Zhongsheng Hou

Citation: Zhen Zhang, Dongbin Zhao. Clique-based cooperative multiagent reinforcement learning using factor graphs. *IEEE/CAA Journal of Automatica Sinica*, 2014, **1**(3): 248−256

Zhen Zhang is with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China and Department of Electric Engineering, College of Automation Engineering, Qingdao University, Qingdao 266071, China (e-mail: zhangzdlut@gmail.com).

Dongbin Zhao is with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: dongbin.zhao@ia.ac.cn).

control[30], adaptive cruise control[31] and pendulum robots control[32]. Crites and Barto[33] used the global reward Q-learning to dispatch a group of elevators. Bazzan et al.[34−35] split traffic signal light agents into groups to reduce the joint action space and proposed a way of coordinating many agents in stochastic games. Kok et al.[36] proposed sparse cooperative Q-learning, where the global Q-value function was decomposed into local Q-value functions, each of which depended only on a small subset of all variables of state and action. Each agent maintained a local Q-value function and updated it with the greedy joint action which was obtained by the max-plus algorithm[37]. However, two problems still remain to be adequately addressed. First, the original sparse Q-learning is not used with a transition function, which usually means a relatively low learning speed. Second, the max-plus algorithm can only deal with local Q-value functions with two variables, which limits the generalization of global function decomposition.

In this paper, our aim is to solve the coordination problem for a class of multiagent systems on tasks which can be decomposed into subtasks. We deal with the problem in two aspects. First, a transition function is estimated and used to update the Q-value function with the aim of reducing the learning time. Second, it is more reasonable to divide agents into cliques, each of which is responsible for a specific subtask. In this case, the global Q-value function is decomposed into the sum of several simpler local Q-value functions which can contain more than two variables. This implies that more flexible decomposition can be considered according to the problem. Such decomposition can be expressed by a factor graph and exploited by the general max-plus algorithm to get the greedy joint action in a distributed manner.

This paper is organized as follows. In Section II, we describe the problem of the coordination for a distributed sensor network (DSN). Section III introduces stochastic games and gives possible popular MARL algorithms for comparison. In Section IV, the clique-based sparse cooperative RL algorithm using factor graphs is proposed. We will show how to update the transition function, how to decompose agents into cliques, and how to use a factor graph to solve the problem. In Section V, the experimental results of various MARL algorithms are presented and compared. Section VI gives the conclusions.

II. Distributed sensor networks

The DSN problem is a distributed optimization problem which was part of the NIPS 2005 benchmarking workshop[38]. It is composed of two arrays of sensors. Fig. 1 shows a DSN with eight sensors, each of which has three actions, i.e., focusing on its left, focusing on its right or not focusing at all. Notice that the action range of corner sensors is not limited to cells. For example, Sensor 0 can focus on its left even if that focus is outside of any cell. There are two targets moving within three cells. Each target has equal probability to move to its left cell, move to its right cell or just stay where it is. The two targets take actions according to the order from left to right. Each cell can be occupied by at most one target at one time. If a target decides to move outside of the three cells or move to a cell which has already been occupied by another target, it will stay where it is. Each target has the maximum energy value (i.e., 3) in the beginning. The energy of a target will be decreased by 1 which is called a hit, if at least three sensors focus on the cell it stays in. If its energy value is 0 which is called a capture, it will get vanished from DSN and do not occupy any cell at all. If all targets are eliminated or 300 steps elapse, an episode is finished[39].

The credit assignment setting follows [39]. Every focus action produces a reward of -1. No focus produces a reward of 0. If a capture is caused by four sensors, only the sensors with three highest indices are rewarded by 10, respectively.

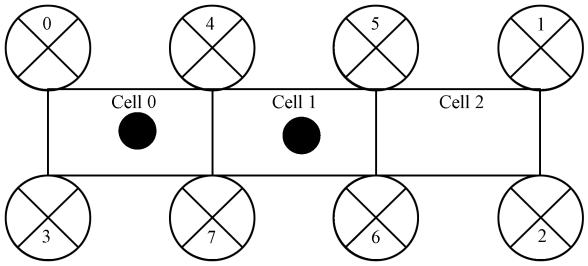


Fig. 1. A distributed sensor network with eight sensors $\otimes$ and two targets $\bullet$.

Notice that sensors do not know whether a hit or a capture has happened but they know the actions of their neighbors. The aim is to obtain as many accumulated rewards as possible in an episode. In this paper, two targets are initially located in two random cells. During each step, the sensors make decisions and act on targets first, causing an intermediate state, then it is the turn for targets to move, transferring to the next state. It is clear that the immediate reward is only dependent on the intermediate state, for they contain the information of whether there is a hit, a capture, focus or no focus. Moreover, we assume that intermediate states can be sensed by all sensors.

In this problem, there are totally $3^8 = 6\,561$ actions and 37 states. In theory, the single agent RL algorithm can learn the optimal strategies for a group of agents if they are regarded as a whole. Nevertheless, there are two issues that make it infeasible for the DSN problem. First, the joint action space grows exponentially as the number of agents increases. Exploring so many state action pairs would be arduous. Second, in the DSN problem, it is impossible for each agent to observe the complete environment state and the actions taken by all the other agents. Next, we introduce several MARL algorithms to solve these problems to some extent.

III. Cooperative multiagent RL

In cooperative MAS, all agents act together to make their environment transit from one state to another, and then they get an immediate global reward. Their objective is to get the maximum accumulated global reward in the long run. Next, we will introduce the theoretical framework that represents such a problem, i.e., stochastic games. Then, several popular MARL algorithms will be introduced and compared with the proposed algorithm.

A. Stochastic Games

A stochastic game[17] is a tuple $<S,\ p,\ A_1, A_2, \cdots, A_n, r_1, r_2, \cdots, r_n>$, where S is a finite or infinite set of environment states, n is the number of agents, A_i is the set of agent i's actions for $i = 1, 2, \cdots, n$, the transition function

$p : S \times A_1 \times A_2 \times \cdots \times A_n \times S \to [0, 1]$ determines the probability of transition from state $\boldsymbol{s}$ to $\boldsymbol{s}'$ under the joint action $\boldsymbol{a}$ of all agents, and $r_i : S \times A_1 \times A_2 \times \cdots \times A_n \times S' \to \mathbf{R}$ is the immediate local reward function of agent i. The immediate global reward of all agents is $r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = \sum_{i=1}^{n} r_i(\boldsymbol{s}, a_i, \boldsymbol{s}')$. We define $A = A_1 \times A_2 \times \cdots \times A_n$ as the set of $\boldsymbol{a}$, a_i as agent i's action and $\boldsymbol{a}_i$ as the joint action of agent i and its neighbors. The learning objective is to maximize the cumulative global reward at each time t, i.e.,

$$R(t) = r(t+1) + \gamma r(t+2) + \cdots + \gamma^T r(t+T+1) = \sum_{k=0}^{T} \gamma^k r(t+k+1), \quad (1)$$

where γ is the discount factor within $[0, 1]$, T is the ending time of an episode, and $r(t+1)$ is the immediate global reward actually received at time $t+1$. The smaller γ is, the more important the rewards at early times become.

A Q-value function represents the expected cumulative global reward for a state $\boldsymbol{s}$ when selecting an action $\boldsymbol{a}$. According to the Bellman equation, the optimal Q-value function can be obtained by

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = \sum_{\boldsymbol{s}'} p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})(r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + r \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}', \boldsymbol{a}')). \quad (2)$$

Once the optimal Q-value function is computed, the greedy joint action for state $\boldsymbol{s}$ is $\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$. For completeness, we give the single agent RL algorithm first.

B. Single Agent RL

We implement single agent RL using value iteration scheme[11], which is a standard way of solving dynamic programming problems. In fact, it is an iterative form of Bellman equation. In value iteration, a transition function is estimated and used to update the Q-value function online as

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \sum_{\boldsymbol{s}'} p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})(r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + rQ(\boldsymbol{s}', \boldsymbol{a}')). \quad (3)$$

The joint action $\boldsymbol{a}$ is selected by the following ε-greedy policy:

$$\boldsymbol{a} = \begin{cases} \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}) \text{ with probability of } 1-\varepsilon, \\ \text{a random action} \in A \text{ with probability of } \varepsilon, \end{cases} \quad (4)$$

where $\varepsilon \in (0, 1)$ is the exploration rate. Next, we introduce several MARL algorithms, where ε-greedy method is also used to balance exploitation and exploration.

C. Independent RL

In independent RL[40], each agent stores and updates a Q-value function of its own. For agent i, the updating rule is

$$Q_i(\boldsymbol{s}, a_i) = Q_i(\boldsymbol{s}, a_i) + \alpha(r_i(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma Q_i(\boldsymbol{s}', a_i') - Q_i(\boldsymbol{s}, a_i)), \quad (5)$$

where $\alpha \in (0, 1)$ is the learning rate that controls the weights of the old Q-value and the estimated error. The dynamics of independent RL have been extensively studied in repeated games[14−16]. Although there is no guarantee for convergence, this algorithm has been applied in some situations.

D. Multiagent Q-learning with Join State Value Approximation

The motivation of the multiagent Q-learning with joint state value approximation (MQVA)[41] is to relieve the dimension disaster problem. Two main processes are described as follows. First, each agent stores a Q-value function and updates it according to

$$Q_i(\boldsymbol{s}, a_i) = Q_i(\boldsymbol{s}, a_i) + \alpha(\gamma(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V(\boldsymbol{s}') - Q_i(\boldsymbol{s}, a_i)). \quad (6)$$

Second, the joint state value function $V(\boldsymbol{s})$ is updated as

$$V(\boldsymbol{s}) = V(\boldsymbol{s}) + \begin{cases} \alpha \Delta V(\boldsymbol{s}), & \text{if } \Delta V(\boldsymbol{s}) > 0, \\ 0, & \text{else}, \end{cases} \quad (7)$$

where $\Delta V(\boldsymbol{s}) = r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V(\boldsymbol{s}') - V(\boldsymbol{s})$. In [41], if a greedy joint action is performed, the joint state value function should also be updated. However, in the DSN problem, we find that updating $V(\boldsymbol{s})$ by (7) can obtain more accumulated rewards with the price of a little more steps used to capture targets. The greedy joint action is composed of each agent′s greedy action.

E. Sparse RL

Sparse RL[36] is based on the assumption that the global Q-value function can be decomposed into local Q-value functions, each of which is dependent on actions of fewer agents. The dependency between agents can be visualised by a coordinated graph (CG)[42]. A CG is a bipartite graph $G = (V, E)$, in which each node $\in V$ represents an agent and each edge $(i, j) \in E$ means agents i and j are neighbors. Once the structure of a CG is settled, it will not change thereafter. The global function can be decomposed in terms of agents or edges. In this subsection, we only introduce agent-based sparse RL and edge based sparse RL with edge updating.

1) Agent-based sparse RL: The global function can be decomposed in terms of agents. Each agent stores a local Q-value function that depends on a small subset of variables of all state and action. Each agent i updates its Q-value function by

$$Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \alpha(r_i(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma Q_i(\boldsymbol{s}_i', \boldsymbol{a}_i') - Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)), \quad (8)$$

where $\boldsymbol{s}_i$ is the state which can be perceived by agent i. Here, we define the global Q-value function as $Q(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=1}^{n} Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)$, the greedy joint action as $\boldsymbol{a} = \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})$, and the joint action of agent i and its neighbors as $\boldsymbol{a}_i$. We can see that the major difference between sparse RL and other MARL algorithms is that it updates local functions using greedy joint actions instead of local greedy actions with the aim of achieving maximum global Q-value at any time[36].

2) Edge-based sparse RL: Another method of decomposition is based on edges. The global function is decomposed into local functions, each of which depends on the joint action of two agents connected by an edge. Its advantage lies in that the computation expense of global function grows linearly with the number of neighbors. One way to assign credit between edges is the edge-updating rule[36], i.e.,

$$Q_{ij}(\boldsymbol{s}_{ij}, a_i, a_j) = Q_{ij}(\boldsymbol{s}_{ij}, a_i, a_j) + \alpha\bigg(\frac{r_i(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')}{|\Gamma(i)|} + \frac{r_j(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')}{|\Gamma(j)|} + \gamma Q_{ij}(\boldsymbol{s}'_{ij}, a'_i, a'_j) - Q_{ij}(\boldsymbol{s}_{ij}, a_i, a_j)\bigg), \tag{9}$$

where $Q(\boldsymbol{s}, \boldsymbol{a}) = \sum_{(i,j)\in E} Q_{ij}(\boldsymbol{s}_{ij}, a_i, a_j)$, $|\Gamma(i)|$ is the number of agent i's neighbors, $\boldsymbol{s}_{ij}$ is the state perceived by agent i and j. The max-plus algorithm is used to derive the greedy joint action[37].

IV. Clique-based sparse RL using factor graphs

So far, we have reviewed several MARL algorithms. Among them, the sparse RL algorithm has a great potential to learn the optimal global Q-value function and the optimal policy in a distributed way. Its advantage over others is that it utilizes the dependency between agents to decompose the global Q-value function. Then the greedy joint action is computed and evaluated in a distributed way while others just evaluate local greedy actions. However, two problems still remain to be adequately addressed. First, the convergence speed of the original sparse RL is slow. Second, the global function can only be decomposed into local functions containing two variables, which might be unreasonable in some cases. Thus in the next section we extend it to the case that local functions contain more than two arguments. We will show that the proposed method reduces the learning time and improves the quality of learned strategies.

A. Sparse RL with a Transition Function

We combine the original sparse RL[36] with a transition function to update the Q-value function as

$$Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \alpha\bigg(\sum_{\boldsymbol{s}'_i \in S_i} p(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i)\big(r_i(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + \gamma Q_i(\boldsymbol{s}'_i, \boldsymbol{a}'_i)\big) - Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)\bigg). \tag{10}$$

The transition function p is updated online by counting method[11]. The pseudo-code of the complete algorithm for agent i is given in Algorithm 1. $n(\boldsymbol{s}_i, \boldsymbol{a}_i)$ represents the number of visited state-action pair $(\boldsymbol{s}_i, \boldsymbol{a}_i)$. $m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i)$ represents the number of visited state-action-state triple $(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i)$.

Algorithm 1. The pseudo-code of model-based sparse RL algorithm.

1: Initialize $Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = 0, n(\boldsymbol{s}_i, \boldsymbol{a}_i) = 0$, for $\{(\boldsymbol{s}_i, \boldsymbol{a}_i)|\boldsymbol{s}_i \in S_i, \boldsymbol{a}_i \in A_i\}$
2: $p(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i) = 0, r_i(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) = 0, m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) = 0$
3: **for** $\{(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i)|\boldsymbol{s}_i \in S_i, \boldsymbol{a}_i \in A_i, \boldsymbol{s}'_i \in S_i\}$ **do**
4: memoryAction($\boldsymbol{s}_i$)=0 for $\{\boldsymbol{s}_i|\boldsymbol{s}_i \in S_i\}$
5: Select an action $\boldsymbol{a}_i$ by some policy (like ε-greedy policy)
6: **end for**
7: **repeat**
8: Execute action $\boldsymbol{a}_i$
9: Observe state $\boldsymbol{s}'_i$ and immediate reward r_i
10: $n(\boldsymbol{s}_i, \boldsymbol{a}_i) = n(\boldsymbol{s}_i, \boldsymbol{a}_i) + 1$
11: $m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) = m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + 1$
12: $p(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i) = \frac{m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i)}{n(\boldsymbol{s}_i, \boldsymbol{a}_i)}$
13: $r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) = r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + \frac{r_i - r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i)}{m(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i)+1}$
14: $\boldsymbol{a}' = \begin{cases} \arg\max_{\boldsymbol{a}'} Q(\boldsymbol{s}', \boldsymbol{a}') & \text{with the probability of } 1-\varepsilon \\ \text{a random action} & \text{with the probability of } \varepsilon \end{cases}$
15: $Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) = Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \alpha(\sum_{\boldsymbol{s}_i \in S_i} p(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i)(r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + \gamma Q_i(\boldsymbol{s}'_i, \boldsymbol{a}'_i)) - Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i))$
16: memoryAction($\boldsymbol{s}_i$) = $\boldsymbol{a}_i$
17: $\boldsymbol{s}_i \to \boldsymbol{s}_i$
18: $\boldsymbol{a}_i \to \boldsymbol{a}_i$
19: **until** the pre-defined number of iterations is reached
20: **return**

The greedy joint action $\boldsymbol{a}$ is acquired by the general max-plus algorithm which will be described later. Notice that when a local function is being updated, it is impossible for agent i to independently acquire the greedy joint action in each state $\boldsymbol{s}'_i$, the only thing it can do is to resort to a table memory action($\boldsymbol{s}'_i$) in which once state $\boldsymbol{s}'_i$ is met, the computed greedy joint action $\boldsymbol{a}'_i$ is stored.

In the DSN problem, the immediate local reward and the next state are determined after the actions are performed. Equation (10) can be simplified as

$$\begin{aligned} Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) &= Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \alpha\bigg(\sum_{\boldsymbol{s}'_i} p(\boldsymbol{s}'_i|\boldsymbol{s}_i, \boldsymbol{a}_i)\big(r_i(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) + \gamma Q_i(\boldsymbol{s}'_i, \boldsymbol{a}'_i)\big) - Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)\bigg) = Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \\ &\alpha\bigg(\sum_{\boldsymbol{s}'_i} p(\boldsymbol{s}'_i|\boldsymbol{s}_i^{inter})\big(r_i + \gamma Q_i(\boldsymbol{s}'_i, \boldsymbol{a}'_i)\big) - Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)\bigg) = \\ &Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i) + \alpha\bigg(r_i + \gamma \sum_{\boldsymbol{s}'_i} p(\boldsymbol{s}'_i|\boldsymbol{s}_i^{inter}) Q_i(\boldsymbol{s}'_i, \boldsymbol{a}'_i) - \\ &Q_i(\boldsymbol{s}_i, \boldsymbol{a}_i)\bigg), \end{aligned} \tag{11}$$

where $\boldsymbol{s}_i^{inter}$ is the intermediate state available for agent i. Thus only the transition function p: $\boldsymbol{s}_i \times \boldsymbol{s}_i^{inter} \to [0, 1]$ needs to be estimated.

B. Clique-based Decomposition

How to assign credits among multiple agents is an important issue in MAS. The simplest approach is to split the global reward equally among agents. This method is called global reward and is widely used in cooperative MAS. It may produce lazy agents because the reward received by an agent does not necessarily depend on its own contribution. Another extreme way is to assign credits to each agent according to its own behavior, which is called local reward. But its final outcome may deviate from the designer's original intention, i.e., cooperation, for there is no explicit mechanism to promote agents to help each other[43].

In this paper, we propose a new method by hybridizing the two methods mentioned above. In fact, our method is very similar to hybrid team learning[43] which has been investigated in genetic programming by Luke[44]. In this kind of learning, agents with the same local interest are clustered into one clique. In each clique, there is one learner in charge of learning and deciding for each agent within the same clique. It receives rewards according to the performance of its clique, stores and updates a local Q-value function which is dependent on the local states sensed by its clique and the joint actions of agents in its clique.

As shown in Fig. 2, eight sensors are split into three cliques. We cluster them in the way that each clique is responsible for hitting and capturing a target in a cell. The topology is stationary thereafter.

Since there are overlaps between cliques, we make the following credit assignment rules. Each clique occupies a cell, for example, Clique 0 occupies Cell 0. In each step, if a sensor does not focus on the cell occupied by its clique, its immediate reward will not be added into its clique. Each clique can only sense what happened in the cell it occupies. Take Clique 1 for example, it can only distinguish four states, i.e., whether there is a target in Cell 1, and how much energy is left (1, 2, or 3).

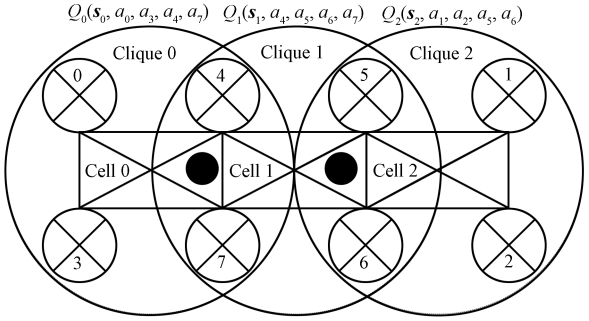


Fig. 2. Clique-based decomposition.

C. Factor Graphs and The General Max-plus Algorithm

The suggestion to use factor graphs comes from [39]. As for a specific state, Q-value function is only dependent on action variables. Thus the global Q-value function is decomposed in terms of cliques as

$$Q(a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7) = Q_0(a_0,a_3,a_4,a_7)+ Q_1(a_4,a_5,a_6,a_7)+Q_2(a_1,a_2,a_5,a_6), \quad (12)$$

where the local Q-value functions Q_0, Q_1, and Q_2 are stored and updated by the learners of the three cliques, respectively. Decomposition (12) can be expressed by a factor graph[45] as shown in Fig. 3(a). A factor graph is a bipartite graph, and it comprises variable nodes and factor nodes. A variable node is visualized as an empty circle to represent a variable. A factor node is visualized as a solid square to represent a factor. For clarity, we use node(a_i) and node(Q_j) to represent the node for variable a_i and the node for function Q_j, respectively. An edge connects node(a_i) and node(Q_j) if and only if a_i is an argument of Q_j. Two nodes are neighbors if and only if there is an edge between them. The decomposition of the global Q-value function in the DSN problem should obey the following rules. On one hand, the factor graph of the sum of the local Q-value functions should be connected, that is, there should be at least one path for any two nodes in the factor graph. Otherwise, there must be standalone agents or cliques. On the other hand, the loops in the factor graph should be as few as possible, because they will cause explosion of messages during the message passing process. Efficient algorithms for solving the maximum of (12) often exploit the possibility of factorizing the global function. The first such algorithm is belief propagation[46] which is used to approximate marginal probability in a Bayesian belief network. Kschischang et al.[45] proposed a general manner of belief propagation in a factor graph which is called sum-product algorithm. The general max-plus algorithm can be obtained from the sum-product algorithm once we use maximization instead of summation and then transform it to log domain. It follows a simple rule that each node keeps sending messages to all of its neighbors until the termination condition is met. There are two types of messages. Let $N(x)$ represent the set of neighbors of node x. The message sent from the variable node node(a_i) to the factor node node(Q_j) is

$$\mu^{t+1}_{\text{node}(a_i)\to\text{node}(Q_j)}(a_i) = \sum_{\text{node}(Q_k)\in N(\text{node}(a_i))\backslash\{\text{node}(Q_j)\}} \mu^{t}_{\text{node}(Q_k)\to\text{node}(a_i)}(a_i), \quad (13)$$

where $N(\text{node}(a_i))\backslash\{\text{node}(Q_j)\}$ denotes the set of nodes which belong to $N(\text{node}(a_i))$ except node(Q_j).

The message μ sent from the factor node node(Q_p) to the variable node node(a_l) is

$$\mu^{t+1}_{\text{node}(Q_p)\to\text{node}(a_l)}(a_l) = \max_{\boldsymbol{a}_p\backslash a_l}\left(Q_p(\boldsymbol{a}_p)+ \sum_{\text{node}(a_k)\in N(\text{node}(Q_p))\backslash\{\text{node}(a_l)\}} \mu^{t}_{\text{node}(a_k)\to\text{node}(Q_p)}(a_k)\right), \quad (14)$$

where $\boldsymbol{a}_p\backslash a_l$ represents the arguments of local function Q_p except a_l. When messages do not change any longer or the predefined number of iterations is reached, the maximum solution can be computed by maximizing messages received by each variable node, respectively, as

$$a_i^* = \arg\max_{a_i} b(a_i) = \arg\max_{a_i} \sum_{\text{node}(Q_k)\in N(\text{node}(a_i))} \mu_{\text{node}(Q_k)\to\text{node}}(a_i). \quad (15)$$

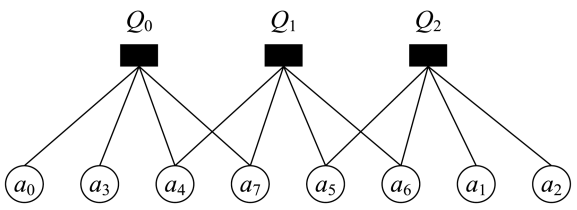
The pseudo-code of the general max-plus algorithm is shown in Algorithm 2. The number of variable nodes and factor nodes are represented by n and m, respectively. In a factor graph, message sending can be sequential or parallel. In one way, messages are sent step by step from leaf nodes to root nodes and then back propagated from root nodes to leaf nodes. In another way, as shown in Algorithm 2, one node does not have to wait for incoming messages before sending them. If a unique a_i exists for maximizing $b(a_i)$, the maximum solution is unique[47]. Under this assumption, exact solution could be obtained by using the general max-plus algorithm in a loop-free factor graph.

Algorithm 2. The pseudo-code of the general max-plus algorithm

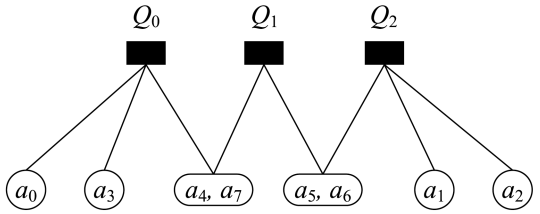
1: Initialize
2: $\mu^0_{\text{node}(a_i)\to\text{node}(Q_j)}(a_i) = 0$ for all $a_i \in A_i$, $\text{node}(Q_j) \in N(\text{node}(a_i))$, $i = 1, 2, \cdots, n$
3: $\mu^0_{\text{node}(Q_p)\to\text{node}(a_l)}(a_l) = 0$ for all $a_l \in A_l$, $\text{node}(a_l) \in N(\text{node}(Q_p))$, $p = 1, 2, \cdots, m$
4: Time $t = 0$
5: **repeat**
6: **for** each variable node $\text{node}(a_i)$ **do**
7: **for** each factor node $\text{node}(Q_j) \in N(\text{node}(a_i))$ **do**
8: Send message for all $a_i \in A_i$
9: $\mu^{t+1}_{\text{node}(a_i)\to\text{node}(Q_j)}(a_i) = \sum_{\text{node}(Q_k)\in N(\text{node}(a_i))\setminus\{\text{node}(Q_j)\}} \mu^t_{\text{node}(Q_k)\to\text{node}(a_i)}(a_i)$
10: **end for**
11: **end for**
12: **for** each factor node $\text{node}(Q_p)$ **do**
13: **for** each variable node $\text{node}(a_l) \in N(\text{node}(Q_p))$ **do**
14: Send message for all $a_l \in A_l$
15: $\mu^{t+1}_{\text{node}(Q_p)\to\text{node}(a_l)}(a_l) = \max_{\boldsymbol{a}_p\setminus a_l}\big(Q_p(\boldsymbol{a}_p) + \sum_{\text{node}(a_k)\in N(\text{node}(Q_p))\setminus\{\text{node}(a_l)\}} \mu^t_{\text{node}(a_k)\to\text{node}(Q_p)}(a_k)\big)$
16: **end for**
17: **end for**
18: $t = t + 1$
19: **until** specified iteration number is reached
20: **for** each variable node $\text{node}(a_i)$ **do**
21: $a_i^* = \arg\max_{a_i} b(a_i) = \arg\max_{a_i} \sum_{\text{node}(Q_k)\in N(\text{node}(a_i))} \mu^t_{\text{node}(Q_k)\to\text{node}(a_i)}(a_i)$
22: **end for**
23: **return**

As shown in Fig. 3(a), the factor graph contains loops. Under these circumstances, the exact solution cannot be guaranteed by using the general max-plus algorithm directly. This is due to the fact that the messages received by a node contain the message sent right from it in an undetermined way, leading to an explosion of messages. This problem can be alleviated by using asynchronous message passing[48] and message passing with damping[49]. The mechanism of message passing is still unclear, nevertheless, the belief propagation like algorithms have shown great success in some situations, for example, the decoding of turbo codes[50].

Here, to get the exact maximum of (12), we break the loops by merging variable nodes $\text{node}(a_5)$, $\text{node}(a_6)$ into one node and $\text{node}(a_4)$, $\text{node}(a_7)$ into another node, as shown in Fig. 3(b). The domain of the emerged variables is $A_{56} = A_5 \times A_6$ and $A_{47} = A_4 \times A_7$.



(a) The factor graph for the sum
$Q_0(a_0, a_3, a_4, a_7) + Q_1(a_4, a_5, a_6, a_7) + Q_2(a_1, a_2, a_5, a_6)$



(b) The factor graph after breaking loops by merging nodes

Fig. 3. The factor graph for clique-based decomposition.

In clique-based decomposition, there might be more than two agents in a clique, which means a local Q-value function may have more than two action variables. In this situation, the maximization problem cannot be tackled directly by using the max-plus algorithm proposed by Kok et al.[37]. Although their techniques can be generalized to local functions with more than two variables by converting the concerned graph to one with only pairwise inter-agent dependencies, we believe that it is more natural and convenient to use factor graphs and the general max-plus algorithm to get the greedy joint action in a distributed manner.

Next, we will test our proposed algorithm as well as the reviewed MARL algorithms on a stochastic game-distributed sensor network.

V. Experiments

In this section, a DSN with eight sensors and two targets shown in Fig. 1 is used as our test-bed for the proposed algorithm and other MARL algorithms.

A. Experiment Settings

We average the results of 50 runs, each of which comprises n_l = 10 000 learning episodes and 5 000 strategy evaluation episodes. In each learning episode, Q-value functions are updated online and actions are selected with ε-greedy policy. Exploration is performed synchronously, which means all learners explore or exploit at each step. To realize synchronization, we set the same random seed for each sensor and activate them simultaneously. In each strategy evaluation episode, Q-value functions are stationary and the greedy joint action is always selected. We only present the average performance of 50 episodes which is an episode block. During learning, the learning rate is decreased linearly with the number of learning episodes as

$$\alpha = \alpha - \alpha_{ini}/n_l,$$

where α_{ini} is the initial learning rate and is set to be 0.7. The exploration rate ε is constant and is set to be 0.2. The discount factor γ is constant and is set to be 0.9. As for the general max-plus algorithm, the maximal iteration number is set to be 10.

For a DSN problem, global states mean the position and energy of all targets. Local states are referred to the local environment states which can be sensed by a clique. For example, the learning agent for Clique 1 can sense and distinguish four different local states, i.e., whether there is a target in Cell 0 and how much energy is left (1, 2, or 3). Agent-based sparse RL, edge-based sparse RL, and the proposed algorithm clique-based sparse RL with a transition function employ only local state information while the others use global states information.

B. Experiment Results

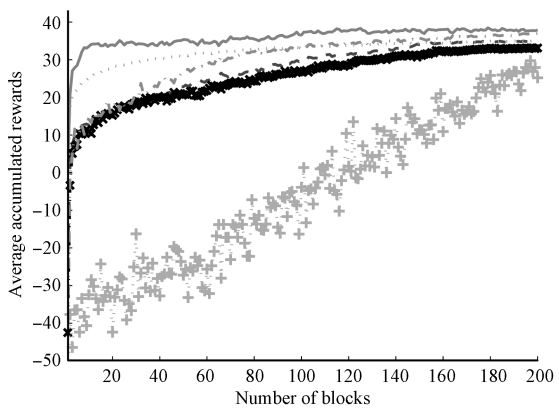
We list the experimental results of single agent RL and MARL algorithms in Fig. 4 for clear comparison.

Single agent RL shows good performance in both average accumulated rewards and steps. However, it obtains the greedy joint action by exhaustion, which means, if there are more sensors in the network, it would probably fail to learn a good strategy in time.

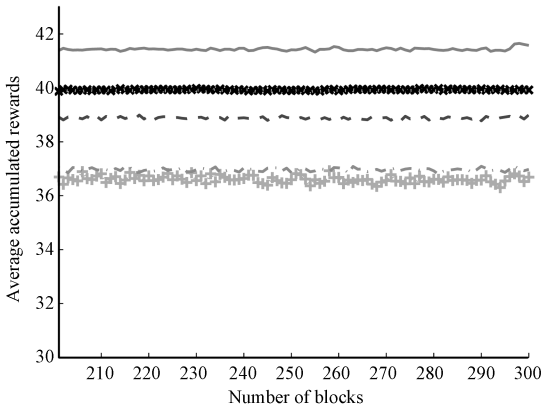
Independent RL takes much more steps to capture targets compared to the other algorithms, so its performance is not shown in Fig. 4(d). We analyze its learnt strategy and find out that sensors do not focus on some states at all. This will not influence the accumulated rewards but do increase the number of steps to capture targets. Independent RL and MQVA obtain less rewards than the other algorithms. We note that the greedy joint action of either independent RL or MQVA is composed of each agent′s greedy action. Although MQVA uses the joint state value function to update each agent′s Q-value function, it does not concern other agents′ influence. Since each agent only takes care of itself, it is hard for them to cooperate well. MQVA and edge-based sparse RL take the second largest number of steps to capture targets.

Agent-based sparse RL shows an average performance. It is a natural way to decompose a task in terms of agents. Yet, it is better to split them into several different cliques in the DSN problem. The average accumulated rewards obtained in this paper are higher than those obtained in [39]. The reason might be the higher learning rate adopted in this paper.

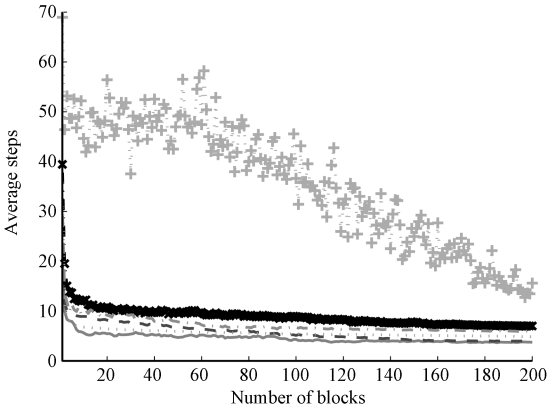
The proposed clique-based sparse RL with a transition function gains the highest average accumulated rewards in minimal steps among all algorithms, that is, more than 41 average accumulated rewards in slightly more than 3 steps, which is close to the best possible performance, i.e., 42 rewards in 3 steps[39]. Moreover, it learns faster than any other algorithm, as shown in Fig. 4.



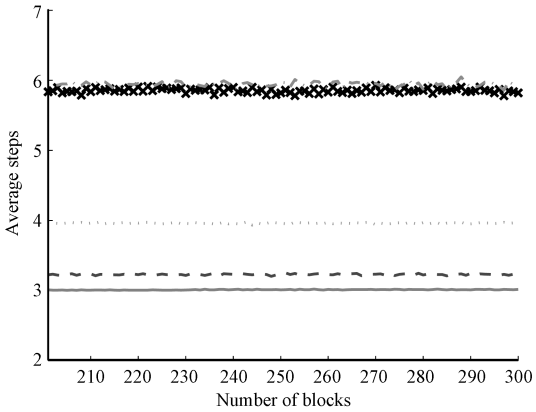
(a) Learning performance on average accumulated rewards per episode



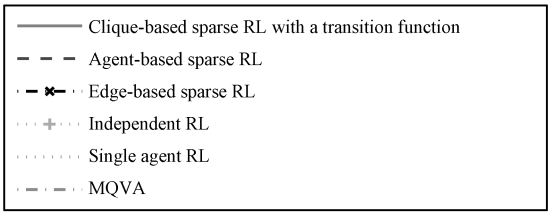
(b) Strategy evaluation on average accumulated rewards per episode



(c) Learning performance on average steps per episode



(d) Strategy evaluation on average steps per episode



(e) Legend

Fig. 4. Average reward and steps using different RL.

In the case of sparse RL and the proposed algorithm, although states are partially observable, they still perform well. We think it is benefited by the characteristics of the DSN problem. The coordination task can be decomposed into subtasks fulfilled by different cliques. This implies that the states needed to fulfill a subtask are more important for a clique. For example, for agents of Clique 1 to hit and capture a target in Cell 0, the states of whether there is a target in Cell 0 and how much energy (1, 2, or 3) is left are much more important than those of whether there is a target in Cell 1 or 2. This might be the reason why sparse RL and the proposed algorithm can perform well in the DSN problem even if only local states are available.

VI. Conclusion

In this paper, we deal with the problem of how to achieve the cooperation in the DSN problem. First, we combine sparse RL with a transition function. Second, we present clique-based decomposition as a method to assign credit among agents. Third, we use the general max-plus algorithm in a factor graph to acquire the greedy joint action. In this way, each agent needs only to sense local environment and communicate with its neighbors. Furthermore, the local Q-value functions can contain more than two variables, which means, more flexible decomposition can be considered according to the problem. Compared with other MARL algorithms, the proposed algorithm gains the best learning performance and produces the best strategies for the DSN problem.

We believe that the best way for the decomposition depends greatly on the problem concerned. In the DSN problem, by splitting sensors into cliques, we formulate a factor graph which can be easily converted to a loop-free one. In more complicated MAS, a factor graph probably has many loops, which makes the convergence and stability of the general max-plus algorithm not hold any more. In the future, we will examine the feasibility and effectiveness of the proposed algorithm in more applications such as coordination of traffic signal lights in an artery and try to solve the problem of factor graphs containing loops.

Acknowledgement

The authors thank Prof. Nikos Vlassis and Jelle. R. Kok for their foundation work on sparse RL. We thank Prof. Derong Liu and Prof. Cesare Alippi for their constructive comments and suggestions on the paper. We thank all the guest editors of the special issue and every member of editorial board for supporting us. We thank Yuzhu Huang , Yujie Dai and Zhongpu Xia for typesetting with Latex.

References

[1] Russell S J, Norvig P, Canny J F, Malik J M, Edwards D D. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs: Prentice Hall, 1995.

[2] Zhao D B, Dai Y J, Zhang Z. Computational intelligence in urban traffic signal control, a survey. *IEEE Transactions on System, Man and Cybernetics Part C: Applications and Reviews*, 2012, **42**(4): 485−494

[3] Li T, Zhao D B, Yi J Q. Adaptive dynamic programming for multi-intersections traffic signal intelligent control. In: Proceedings of the 11th IEEE International Conference on Intelligent Transportation Systems. Beijing, China: IEEE, 2008. 286−291

[4] Zhao D B, Bai X R, Wang F Y, Xu J, Yu W S. DHP for coordinated freeway ramp metering. *IEEE Transactions on Intelligent Transportation Systems*, 2011, **12**(4): 990−999

[5] Vlassis N. A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2007, **1**(1): 1−71

[6] Khan Z, Glisic S, DaSilva L A, Lehtomaki J. Modeling the dynamics of coalition formation games for cooperative spectrum sharing in an interference channel. *IEEE Transactions on Computational Intelligence and AI in Games*, 2011, **3**(1): 17−30

[7] Zhao D B, Zhang Z, Dai Y J. Self-teaching adaptive dynamic programming for Go-Moku. *Neurocomputing*, 2012, **78**(1): 23−29

[8] Tan C H, Tan K C, Tay A. Dynamic game difficulty scaling using adaptive behavior-based AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 2011, **3**(4): 289−301

[9] Zeng L, Hu G D. Consensus of linear multi-agent systems with communication and input delays. *Acta Automatica Sinica*, 2013, **39**(7): 1133−1140

[10] Li T, Fu M Y, Xie L H, Zhang J F. Distributed consensus with limited communication data rate. *IEEE Transactions on Automatic Control*, 2011, **56**(2): 279−292

[11] Sutton R S, Barto A G. *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998

[12] Wiering M A. Multiagent reinforcement learning for traffic light control. In: Proceedings of the 17th International Conference on Machine Learning. Stanford University, US: Morgan Kaufmann, 2000. 1151−1158

[13] Claus C, Boutilier C. The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings of the 15th National Conference on Artificial Intelligence and 10th Conference on Innovative Applications of Artificial Intelligence. Menlo Park, CA: AAAI Press/MIT Press, 1998. 746−752

[14] Tuyls K, Nowé A. Evolutionary game theory and multi-agent reinforcement learning. *The Knowledge Engineering Review*, 2005, **20**(1): 63−90

[15] Gomes E R, Kowalczyk R. Dynamic analysis of multiagent Q-learning with ε-greedy exploration. In: Proceedings of the 26th International Conference on Machine Learning. New York, USA: ACM, 2009. 369−376

[16] Kianercy A, Galstyan A. Dynamics of Boltzmann Q-learning in two-player two-action games. *Physical Review E*, 2012, **85**(4): 1145−1154

[17] Littman M L. Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the 11th International Conference on Machine Learning. New Brunswick, US: Morgan Kaufmann, 1994. 157−163

[18] Owen G. *Game Theory* (*Second Edition*). Orlando, Florida: Academic Press, 1982.

[19] Littman M L. Friend-or-foe Q-learning in general-sum games. In: Proceedings of the 18th International Conference on Machine Learning. San Francisco, CA: Morgan Kaufmann, 2001. 322−328

[20] Hu J L, Wellman M P. Multiagent reinforcement learning: theoretical framework and an algorithm. In: Proceedings of the 15th International Conference on Machine Learning. Madison, Wisconsin, US: Morgan Kaufmann, 1998. 242−250

[21] Singh S, Kearns M, Mansour Y. Nash convergence of gradient dynamics in general-sum games. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence. Stanford University, Stanford, California, US: Morgan Kaufmann, 2000. 541−548

[22] Bowling M, Veloso M. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 2002, **136**(2): 215−250

[23] Zhang Hua-Guang, Zhang Xin, Luo Yan-Hong, Yang Jun. An overview

of research on adaptive dynamic programming. *Acta Automatica Sinica*, 2013, **39**(4): 303−311 (in Chinese)

[24] Werbos P J. A menu of designs for reinforcement learning over time. *Neural Networks for Control.* Cambridge: MIT Press, 1990

[25] Liu D R, Wang D, Zhao D B, Wei Q L, Jin N. Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming. *IEEE Transactions on Automation Science and Engineering*, 2012, **9**(3): 628−634

[26] Huang Y Z, Liu D R. Neural-network-based optimal tracking control scheme for a class of unknown discrete-time nonlinear systems using iterative ADP algorithm. *Neurocomputing*, 2014, **125**: 46−56

[27] Song Rui-Zhuo, Xiao Wen-Dong, Sun Chang-Yin. Optimal tracking control for a class of unknown discrete-time systems with actuator saturation via data-based ADP algorithm. *Acta Automatica Sinica*, 2013, **39**(9): 1413−1420 (in Chinese)

[28] Wang D, Liu D R. Neuro-optimal control for a class of unknown nonlinear dynamic systems using SN-DHP technique. *Neurocomputing*, 2013, **121**(9) 218−225

[29] Zhang Ji-Lie, Zhang Hua-Guang, Luo Yan-Hong, Liang Hong-Jing. Nearly optimal control scheme using adaptive dynamic programming based on generalized fuzzy hyperbolic model. *Acta Automatica Sinica*, 2013, **39**(2): 142−148 (in Chinese)

[30] Liu D R, Javaherian H, Kovalenko O, Huang T. Adaptive critic learning techniques for engine torque and air-fuel ratio control. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 2008, **38**(4): 988−993

[31] Zhao D B, Hu Z H, Xia Z P, Alippi C, Zhu YH, Wang D. Full range adaptive cruise control based on supervised adaptive dynamic programming. *Neurocomputing*, 2014, **125**: 57−67

[32] Zhao D B, Yi J Q, Liu D R. Particle swarn optimized adaptive dynamic programming. In: Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. Honolulu, Hawaiian Islands, US: IEEE, 2007. 32−37

[33] Crites R H, Barto A G. Improving elevator performance using reinforcement learning. In: Proceedings of Advances in Neural Information Processing Systems 8. Denver, US: MIT Press, 1996. 1017−1023

[34] Bazzan A L, de Oliveira D, de Silva B C. Learning in groups of traffic signals. *Engineering Applications of Artificial Intelligence*, 2010, **23**(4): 560−568

[35] Bazzan A L. Coordinating many agents in stochastic games. In: Proceedings of the 2012 International Joint Conference on Neural Networks. Brisbane, Australia: IEEE, 2012. 1−8

[36] Kok J R, Vlassis N. Sparse cooperative Q-learning. In: Proceedings of the 21st International Conference on Machine Learning. USA: ACM, 2004. 481−488

[37] Kok J R, Vlassis N. Using the max-plus algorithm for multiagent decision making in coordination graphs. In: Proceedings of Robot Soccer World Cup IX, Lecture Notes in Computer Science. Berlin: Springer, 2005. 1−12

[38] Syed A, Koenig S, Tambe M. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems. Utrecht, The Netherlands: ACM, 2005. 1041−1048

[39] Kok J R, Vlassis N. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 2006, **7**(S1): 1789−1828

[40] Schneider J, Wong W K, Moore A, Riedmiller M. Distributed value functions. In: Proceedings of the 16th International Conference on Machine Learning. San Francisco, CA: Morgan Kaufmann, 1999. 371−378

[41] Chen G, Cao W H, Chen X, Wu M. Multi-agent Q-learning with joint state value approximation. In: Proceedings of the 30th Chinese Control Conference. Yantai, China: IEEE, 2011. 4878−4882

[42] Guestrin C, Lagoudakis M G, Parr R. Coordinated reinforcement learning. In: Proceedings of the 19th International Conference on Machine Learning. Sydney, Australia: Morgan Kaufmann, 2002. 227−234

[43] Panait L, Luke S. Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-Agent Systems*, 2005, **11**(3): 387−434

[44] Luke S. Genetic programming produced competitive soccer softbot teams for RoboCup97. In: Proceedings of the 3rd Annual Conference on Genetic Programming. Madison, Wisconsin, US: Morgan Kaufmann, 1998. 214−222

[45] Kschischang F R, Frey B J, Loeliger H A. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 2001, **47**(2): 498−519

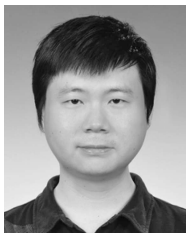
[46] Pearl J. *Probabilistic Reasoning in Intelligent Systems.* San Mateo: Morgan Kaufman, 1988

[47] Weiss Y, Freeman W T. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 2001, **47**(2): 736−744

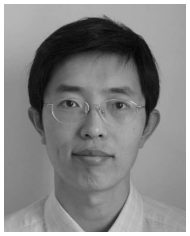
[48] Lan X, Roth S, Huttenlocher D, Black M. Efficient belief propagation with learned higher-order Markov random fields. In: Proceedings of the 2006 European Conference on Computer Vision. Berlin: Springer, 2006. 269−282

[49] Som P, Chockalingam A. Damped belief propagation based near-optimal equalization of severely delay-spread UWB MIMO-ISI channels. In: Proceedings of the 2010 IEEE International Conference on Communications. Cape Town, South Africa: IEEE, 2010. 1−5

[50] Frey B J, Kschischang F R, Conference A. Probability propagation and iterative decoding. In: Proceedings of the 34th Annual Allerton Conference on Communication Control and Computing. Illinois, US: IEEE, 1996. 1−4



Zhen Zhang Received the Ph. D. degree from the Institute of Automation, Chinese Academy of Sciences, China in 2013. He received the M. S. degree in control theory and control engineering from Dalian University of Technology, China in 2009, and the B. S. degree from China University of Petroleum, China in 2006. He is currently a lecturer in the Department of Electric Engineering, College of Automation Engineering, Qingdao University, China. His main research interest covers the application of reinforcement learning and neural networks to urban traffic signal control.



Dongbin Zhao Received the B. S., M. S., and Ph. D. degrees from the Harbin Institute of Technology, China, in August 1994, August 1996, and April 2000, respectively. He was a postdoctoral fellow in Tsinghua University, China, from May 2000 to January 2002. He is currently a professor with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, China. He has published one book and more than 30 international journal papers. His current research interest covers the area of computational intelligence, adaptive dynamic programming, robotics, intelligent transportation systems, and process simulation. Corresponding author of this paper.