12-2007

Self-organizing neural architectures and cooperative learning in a multiagent environment

Dan XIAO

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Self-Organizing Neural Architectures and Cooperative Learning in a Multiagent Environment

Article *in* IEEE TRANSACTIONS ON CYBERNETICS · January 2008

DOI: 10.1109/TSMCB.2007.907040 · Source: PubMed

2 authors, including:

Ah-Hwee Tan
Nanyang Technological University
234 PUBLICATIONS **4,059** CITATIONS

SEE PROFILE

Self-Organizing Neural Architectures and Cooperative Learning in a Multiagent Environment

Dan Xiao and Ah-Hwee Tan, *Senior Member, IEEE*

***Abstract*—Temporal-Difference–Fusion Architecture for Learning, Cognition, and Navigation (TD-FALCON) is a generalization of adaptive resonance theory (a class of self-organizing neural networks) that incorporates TD methods for real-time reinforcement learning. In this paper, we investigate how a team of TD-FALCON networks may cooperate to learn and function in a dynamic multiagent environment based on minefield navigation and a predator/prey pursuit tasks. Experiments on the navigation task demonstrate that TD-FALCON agent teams are able to adapt and function well in a multiagent environment without an explicit mechanism of collaboration. In comparison, traditional Q-learning agents using gradient-descent-based feedforward neural networks, trained with the standard backpropagation and the resilient-propagation (RPROP) algorithms, produce a significantly poorer level of performance. For the predator/prey pursuit task, we experiment with various cooperative strategies and find that a combination of a high-level compressed state representation and a hybrid reward function produces the best results. Using the same cooperative strategy, the TD-FALCON team also outperforms the RPROP-based reinforcement learners in terms of both task completion rate and learning efficiency.**

***Index Terms*—Multiagent cooperative learning, reinforcement learning (RL), self-organizing neural architectures.**

I. Introduction

A KEY challenge of autonomous agents is to function and adapt by themselves in a complex and dynamic environment. In a multiagent system, the adaptation task is particularly challenging. First of all, an agent is affected by the actions of the other agents, and its action also affects the environment and the other agents. Second, as agents continue to learn, their behaviors change over time, and the environment becomes highly dynamic. An agent must therefore be able to predict the actions or to model the reasoning processes of the others in some way.

A natural approach to both single-agent and multiagent learning is reinforcement learning (RL), as it enables an autonomous agent to learn through interaction with the environment, based on the consequences of actions, instead of explicit teaching. Following the framework of a Markov decision process (MDP) [1], an RL agent typically operates in a sense, act, and learn cycle. In each cycle, the agent first obtains sensory input from its environment representing the current state $(\mathbf{S})$. Depending on the current state and its knowledge and goals, the agent selects and performs the most appropriate action $(\mathbf{A})$. Upon receiving the feedback in terms of rewards $(\mathbf{R})$ from the environment, the agent learns to adjust its behavior in the motivation of receiving positive rewards in the future. In situations where the current state $\mathbf{S}$ is based on observations made by the agent instead of absolute states, the problem becomes a partially observable MDP (POMDP) [2].

This paper investigates how a class of self-organizing neural networks, known as Fusion Architecture for Learning, Cognition, and Navigation (FALCON) [3], can function as autonomous agents that learn and function in a team through their interaction with the environment. FALCON learns multimodal mappings simultaneously across multiple-pattern channels, involving states, actions, and rewards, in an online and incremental manner. A specific class of FALCON models, called temporal-difference–FALCON (TD-FALCON) [4], [5], learns the value functions of the state-action space estimated through TD algorithms [1]. A key advantage of TD methods is that they can be used for multiple-step prediction problems, in which the information on the correctness of an action can only be available after several steps into the future. The learned value functions are then used to determine the optimal actions based on an action selection policy. To achieve high performance, we adopt a hybrid action selection policy that initially favors exploration and gradually leans toward exploitation in the later stage of the learning process.

To investigate how TD-FALCON may operate in a multiagent setting, this paper conducts empirical studies based on a multiagent minefield navigation task and a multipredators/prey pursuit task. The former involves a number of autonomous vehicles (AVs) learning to navigate through obstacles to reach a stationary target (goal) within a specified number of steps [6]. The experimental results show that, using the TD-FALCON model, the AVs adapt very well and learn to perform the task rapidly despite the presence and the interference of the other agents. In comparison, the same level of performance is not attainable by traditional Q-learning agents based on multilayer feedforward neural networks trained with the backpropagation (BP) algorithm [7] and the resilient-propagation (RPROP) algorithm [8], [9] as the function approximator.

The pursuit task involves multiple agents pursuing a moving target. This task calls for an explicit mechanism of cooperation, as it requires the agents to surround the prey from all directions [10]. Our experiments show that the agents do not need to have in-depth knowledge of the other agents' underlying models to accomplish the task successfully. Observing the

Manuscript received October 19, 2006; revised June 7, 2007. This work was supported in part by the Intelligent Systems Centre, Nanyang Technological University, under a research grant. This paper was recommended by Associate Editor P. De Wilde.

The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore (e-mail: xiao0002@ntu.edu.sg; asahtan@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/TSMCB.2007.907040

environment and learning in a reactive manner enable the agents to predict the possible actions of other agents and potential outcomes of their own. In addition, our experiments on pursuit games demonstrate that appropriate state representation and cooperative strategies play a critical role in producing superior performance while maintaining efficiency and scalability.

The rest of this paper is organized as follows. Section II gives a retrospect of the state of the art on multiagent learning. Section III provides a summary of the FALCON architecture together with the associated value-function learning and prediction algorithms. Section IV presents the TD-FALCON algorithms, including the action selection policy and the value-function-estimation mechanism. Sections V and VI report our experiments on the multiagent minefield navigation task and the multipredators/prey pursuit game, respectively. Section VII concludes and highlights the future work.

II. Related Work

The focus of this paper is on cooperative multiagent learning, whereby multiple agents cooperate to solve a given task jointly and/or to maximize certain utility function through their interactions [11]. Generally, there are two key approaches to cooperative learning, namely, stochastic search and RL. Stochastic search operates by searching through the solution space and selecting from randomly generated candidate solutions, as in the field of evolutionary computation (EC). Typically, a fitness-oriented procedure is used to refine the multiple agents' behaviors [11], [12]. EC is computationally intensive, and thus, it is not commonly used in real-time applications [13]. This paper, on the other hand, adopts the approach of RL, a paradigm wherein an agent perceives the current state of the environment and takes actions according to the rewards or penalties received in a real-time manner [14], [15].

A. Multiagent RL

Classical RL methods involve learning one or both of the following functions, namely, policy function, which maps each state to a desired action, and value function, which associates each pair of state and action to a utility value. For learning value function, a popularly used method is Q-learning [16], which is a temporal-difference method to estimate the accumulative future rewards (or costs) of performing an action in a given state. Under standard Q-learning, an agent must establish a table to store each tuple of state, action, and Q values. Such a requirement causes a scalability problem for large and/or continuous state and action spaces. In a multiagent system, an agent needs to keep track of its environment, as well as the other agents, aggravating the scalability problem.

To tackle the scalability problem of multiagent Q-learning, many approaches based on function approximation have been proposed. Kose *et al.* [17] used cerebellar-model articulation controller for function approximation and state generalization in robot soccer games because of its efficiency in learning and operation. Kononen [18] proposed a gradient-based method, which is a combination of value-function approximation and direct-policy gradient with the value-and-policy-search framework. However, as Yang and Gu [19] pointed out, multiagent Q-learning with function approximation can be very unstable. Since there is no way to store all the patterns, learning new patterns may erase the previously learned knowledge. In addition, most function-approximation methods rely on slow and iterative learning, which is not effective to cope in a real-time environment.

The TD-FALCON system, as described in this paper, can also be viewed as a function-approximation approach to Q-learning. However, by inheriting the fast and stable learning characteristics from adaptive resonance theory (ART) models, TD-FALCON is able to learn the Q-value function in a much faster pace, while retaining the learning stability.

B. Multiagent Cooperative Strategies

Multiagent cooperative strategies can be broadly classified into team learning and concurrent learning [11]. Team learning involves a single learner to carry out the learning behaviors of all the agents. The centralized team learning algorithm requires all resources and information to be collected by a single learner, and thus, the communication load is increased in an inherently distributed system [20].

Using concurrent learning, each agent is equipped with a learner to do the learning simultaneously [21]. Concurrent learning generally involves three main considerations, namely, reward assignment [22], dynamics of learning [23], and modeling of other agents [24].

Reward assignment can be in the forms of global rewards [25], local rewards [26], and/or observational reinforcement rewards [27]. Using global rewards, a team payoff is distributed equally among all the learners. Using local reward, the reward of each agent is purely based on its own behavior. Using observational reinforcement reward, an agent gains its rewards by observing and imitating other agents' behaviors.

In terms of learning dynamics, there are generally two scenarios, namely, fully cooperative [28] and general-sum game [29]. In a fully cooperative scenario, the rewards received by agents are correlated, and so, increasing one agent's reward implies increasing other agents' rewards. In the case of the general-sum game, increasing an agent's reward does not necessarily result in increasing the reward of the whole team. Such a scenario may lead to highly noncooperative situations.

In terms of modeling other agents, there are zero-level modeling agents, one-level modeling agents, two-level modeling agents, and so on [30]. Generally, a zero-level agent does not consider whether any of its teammates is performing any learning activity. A one-level agent models its teammates as zero-level agents, and in general, an N-level agent models its teammates as $(N-1)$-level agents. Although higher level agent models appear to be more powerful, prior studies [31]–[33] have found that the simplest zero-level agents can have better performance than one-level and two-level agents.

Existing works on multiagent systems have explored the various combinations of reward assignment, dynamics of learning, and agent modeling. For example, Littman [34] proposed a minimax Q-learning algorithm to update the V values in two-player zero-sum stochastic games. Tan [35] showed that

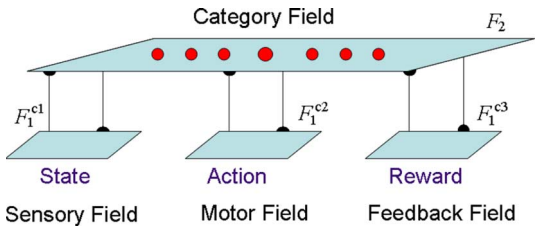

Fig. 1. FALCON architecture.

cooperative agents using the minimax Q-learning algorithm can significantly outperform independent agents. However, the minimax Q-learning is limited to exactly two agents with diametrically opposed goals. For general sum stochastic games, Hu and Wellman [36] proposed the Nash–Q algorithm to update the V values based on some Nash equilibrium. Unfortunately, the solution is restricted to only two players and to the condition that the other agent learns the same Nash equilibrium. Agogino and Tumer [37] proposed QUICR-learning to divide a global reward into many agent-specific rewards in a large-scale multiagent system. The proposal has two properties: 1) agents increasing their agent-specific rewards tend to increase the global reward and 2) an agent's action has a great influence on its agent-specific reward. The disadvantage is that QUICR is based on TD(0), which depends on immediate rewards.

The TD-FALCON systems presented in this paper employ a concurrent-learning strategy. In terms of reward assignment, we employ a hybrid reward mechanism in the predator/prey pursuit task, combining individual and team payoffs. In terms of dynamics of learning, we focus on a fully cooperative scenario, in which all agents cooperate to maximize their gains. In the aspect of modeling other agents, TD-FALCON systems adopt the zero-level model, meaning that an agent regards other agents' behaviors as part of the dynamic environment.

III. FALCON Architecture

The FALCON is an extension of ART networks [38], [39] for learning multimodal pattern mappings across multiple-input channels. For RL, the FALCON makes use of a three-channel architecture, consisting of a sensory field F_1^{c1} for representing the current state, an action field F_1^{c2} for representing the available actions, a reward field F_1^{c3} for representing the values of the feedback received from the environment, and a cognitive field F_2^c for learning cognitive nodes, each of which encodes a relation among the patterns in the three input channels (Fig. 1). We describe how the FALCON can be used to predict and learn value functions for RL as follows.

Input vectors: Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where s_i indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where a_i indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ and $\bar{r}$ (the complement of r) is given by $\bar{r} = 1 - r$. Complement coding serves to normalize the magnitude of the input vectors and has been found effective in ART systems in preventing the code proliferation problem [40].

Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector for $k = 1$ to 3. Let $\mathbf{y}^c$ denote the F_2^c activity vector.

Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2^c for learning the input patterns in F_1^{ck}. Initially, all F_2^c nodes are uncommitted, and the weight vectors contain all ones.

Parameters: The FALCON's dynamics is determined by choice parameters $\alpha^{ck} > 0$ for $k = 1$ to 3; learning-rate parameters $\beta^{ck} \in [0, 1]$ for $k = 1$ to 3; contribution parameters $\gamma^{ck} \in [0, 1]$ for $k = 1$ to 3, where $\sum_{k=1}^{3} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0, 1]$ for $k = 1$ to 3.

The dynamics of FALCON for learning and predicting value functions, based on fuzzy ART operations [41], is described in the following sections.

A. Predicting Value Functions

In the predicting mode, FALCON receives input values in one or more fields and predicts the values for the remaining fields. Upon input presentation, the input fields receiving values are initialized to their respective input vectors. Input fields not receiving values are initialized to $\mathbf{N}$, where $N_i = 1$ for all i. For predicting value functions, only the state and action vectors are presented to FALCON. Therefore, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{N}$.

The predicting process of FALCON consists of three key steps, namely, code activation, code competition, and activity readout, which are described as follows.

Code activation: A bottom-up propagation process first takes place in which the activities (known as choice-function values) of the cognitive nodes in the F_2^c field are computed. Given the activity vectors $\mathbf{x}^{c1}, \ldots, \mathbf{x}^{c3}$, the choice function T_j^c of each F_2^c node j is computed as follows:

$$T_j^c = \sum_{k=1}^{3} \gamma^{ck} \frac{\left|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}\right|}{\alpha^{ck} + \left|\mathbf{w}_j^{ck}\right|} \tag{1}$$

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv \min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$.

Code competition: A code competition process follows under which the F_2^c node with the highest choice function value is identified. The system is said to make a choice when, at most, one F_2^c node can become active after code competition. The winner is indexed at J, where

$$T_J^c = \max\left\{T_j^c \ : \ \text{for all } F_2^c \text{ node } j\right\}. \tag{2}$$

When a category choice is made at node J, $y_j^c = 1$ and $y_j^c = 0$ for all $j \neq J$. This indicates a winner-take-all strategy.

Activity readout: The chosen F_2^c node J performs a readout of its weight vectors to the input fields F_1^{ck} as

$$\mathbf{x}^{ck(\text{new})} = \mathbf{x}^{ck(\text{old})} \wedge \mathbf{w}_J^{ck}. \tag{3}$$

Finally, the reward vector $\mathbf{R}$ associated with the input state vector $\mathbf{S}$ and the action vector $\mathbf{A}$ is given by $\mathbf{R} = \mathbf{x}^{c3}$.

B. Learning Value Functions

For learning value functions, the state, action, and reward vectors are presented simultaneously to the FALCON. Therefore, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R}$. Under the learning mode, the FALCON performs code activation and code competition (as described in the previous section) to select a winner J based on the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$. To complete the learning process, template matching and template learning are performed as described below.

Template matching: Before code J can be used for learning, a template-matching process checks whether the weight templates of code J are sufficiently close to their respective input patterns. Specifically, a resonance occurs if, for each channel k, the match function m_j^{ck} of the chosen code J meets its vigilance criterion

$$m_J^{ck} = \frac{\left|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}\right|}{\left|\mathbf{x}^{ck}\right|} \geq \rho^{ck}. \tag{4}$$

When a resonance occurs, the template learning ensues, as defined as follows. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T_j^c is reset to -1 during the input presentation. The search process then continues to select another F_2^c node J until a resonance is achieved.

Template learning: Once a node J is selected for firing, for each channel k, the weight vector $\mathbf{w}_j^{ck}$ is modified by the following learning rule:

$$\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}\left(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})}\right). \tag{5}$$

For an uncommitted node J, the learning rate β^{ck} is typically set to one. For committed nodes, β^{ck} can remain as one for fast learning or below one for slow learning in a noisy environment.

Node creation: Our implementation of FALCON maintains one uncommitted node in the F_2^c field at any one time. When the uncommitted node is selected for learning, it becomes committed and a new uncommitted node is added to the F_2^c field. FALCON, thus, expands its network architecture dynamically in response to the input patterns.

IV. Temporal-Difference–Fusion Architecture for Learning, Cognition, and Navigation

The general sense–act–learn algorithm of TD-FALCON is summarized in Table I. Given the current state s and a set of available actions $\mathcal{A}$, the FALCON network is used to predict the value of performing each available action. The value functions are then processed by an action selection strategy (also known as policy) to select an action. Upon receiving a feedback (if any) from the environment after performing the action, a TD formula is used to estimate the value of the next state. The value is then used as the teaching signal for FALCON to learn the association from the current state and the chosen action to the estimated value.

TABLE I
Generic Dynamics of the TD-FALCON

1. Initialize the FALCON network.
2. Given the current state s, for each available action a in the action set $\mathcal{A}$, predict the value of the action $Q(s,a)$ by presenting the corresponding state and action vectors $\mathbf{S}$ and $\mathbf{A}$ to FALCON.
3. Based on the value functions computed, select an action a from $\mathcal{A}$ following an action selection policy.
4. Perform the action a, observe the next state s', and receive a reward r (if any) from the environment.
5. Estimate the value function $Q(s,a)$ following a TD formula given by $\Delta Q(s,a) = \alpha \text{TD}_{err}$.
6. Present the corresponding state, action, and reward (Q-value) vectors ($\mathbf{S}$, $\mathbf{A}$, and $\mathbf{R}$) to FALCON for learning.
7. Update the current state by s=s'.
8. Repeat from Step 2 until s is a terminal state.

A. Action Selection Policy

To achieve a balance between exploration and exploitation, we adopt an ϵ-greedy strategy, which selects an action of the highest value with the probability of $1 - \epsilon$, where ϵ is a constant between zero and one, or takes a random action otherwise [42]. With a fixed ϵ value, the agent always explores the environment with a constant level of randomness. But in practice, it is beneficial to have a higher ϵ value in the initial stage to encourage the exploration of new paths and a lower ϵ value in the later stage to optimize the performance by exploiting familiar paths. A decayed ϵ-greedy policy is thus proposed to gradually reduce the value of ϵ over time. The decay rate d_ϵ is, typically, inversely proportional to the complexity of the environment as problems with a larger input and action space require a longer time to explore.

B. Value-Function Estimation

One key component of TD-FALCON (Step 5) is the iterative estimation of the value function $Q(s,a)$ using a temporal-difference equation

$$\Delta Q(s,a) = \alpha \text{TD}_{\text{err}} \tag{6}$$

where $\alpha \in [0,1]$ is the learning parameter, and TD_{err} is a function of the current Q value predicted by the FALCON, and the Q value newly computed by the TD formula. Using the Q-learning rule, the temporal-error term is computed by

$$\text{TD}_{\text{err}} = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{7}$$

where r is the immediate-reward value, $\gamma \in [0,1]$ is the discount parameter, and $\max_{a'} Q(s',a')$ denotes the maximum estimated value of the next state s'. The update rule is applied to all states that the agent goes through. With value iteration, the value function $Q(s,a)$ is expected to track and converge to $r + \gamma \max_{a'} Q(s',a')$ (of which the value is also moving) over time. To ensure that all input values are bounded between zero

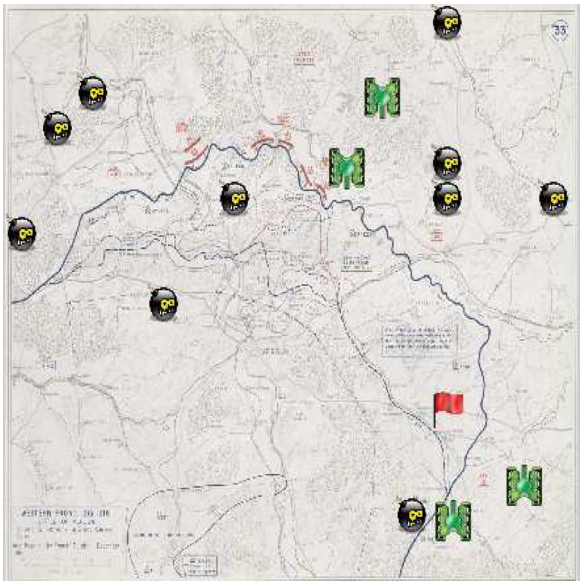

Fig. 2. Multiagent minefield navigation simulator.

and one, a scaling term is incorporated to form the bounded Q-learning rule, which is given by

$$\Delta Q(s,a) = \alpha \mathrm{TD}_{\mathrm{err}} \left(1 - Q(s,a)\right). \tag{8}$$

The ΔQ term is then used to compute a new Q value, which is given by

$$Q^{(\mathrm{new})}(s,a) = Q(s,a) + \Delta Q \tag{9}$$

which, in turn, is encoded into the reward vector for the FALCON as

$$\mathbf{R} = \left(Q^{(\mathrm{new})}(s,a), 1 - Q^{(\mathrm{new})}(s,a)\right). \tag{10}$$

V. Minefield Navigation Task

Minefield navigation has been used in studies of cognitive models [6], [43] as an evaluation task. In the minefield navigation task, a number of AVs navigate through a minefield to a randomly selected target position (Fig. 2). In each trial, the AVs start at randomly chosen positions in the field. The objective of the game is for the AVs to reach the target position in a specified time frame without hitting a mine or colliding with each other. During a trial, the mines remain stationary. A trial ends when each individual AV reaches the target (success), hits a mine (failure), collides with each other (failure), or runs out of time (failure).

A. State Representation

Minefield navigation with mine avoidance is not a trivial task. As the configuration of the minefield is generated randomly and changes over trials, an agent needs to learn strategies that can be carried across experiments. In addition, the agent has a rather weak sensory capability with only a 180° forward view based on five sonar sensors. For each direction i, the sonar signal is measured by $s_i = 1/(1 + d_i)$, where d_i is the distance to an obstacle (that can be either a mine or the boundary of the minefield) in the ith direction. Other input elements of the sensory (state) vector include the range and the bearing of the target from the current position. By using observations instead of physical locations, our formulation of the minefield navigation problem is based on the POMDP. In each step, the system can choose one of the five possible actions, namely, moving left, moving diagonally left, moving straight ahead, moving diagonally right, and moving right.

In a multiagent environment, the minefield navigation task becomes more challenging. First, agents have to avoid collisions with each other as a collision results in failures of both colliding agents. Second, as agents are nonstationary, an agent needs to predict the movement of its neighboring agents to avoid collisions. For the purpose of monitoring the other agents, each agent adds a separate set of five sonar sensors to its sensory representation, because agents and mines have different characteristics and should be tracked separately. Without the additional sonar sensors, our initial experiments produce poor results.

The complexity of learning in the multiagent minefield navigation problem is determined by the dimension of the sensory(state) and action space. The state–action space is given by $\mathbf{S}^{10} \times \mathbf{A} \times \mathbf{B}$, where $\mathbf{S} = [0, 1]$ is the value range of the ten sonar signals, $\mathbf{A}$ is the set of available actions, and $\mathbf{B} = 0, 1, \ldots, 7$ is the set of possible target bearings. With a continuous state–action space, traditional RL systems would have a scalability problem. Even if the sonar signals are simply binary, there are still at least $2^{10} * 5 * 8$ (approximately 40 000) possible combinations of states and actions.

B. Reward Scheme

Our initial experiments are based on a 16 × 16 minefield containing ten mines. In each trial, every individual AV repeats the cycles of sense, act, and learn, until it reaches the target, hits a mine, collides with another AV, or exceeds 30 sense–act–learn cycles. A reward of 1 is given when the AV reaches the target, and a reward of 0 is given when the AV hits a mine or collides with another AV. Under the immediate reward scheme, at each step of the trial, an immediate reward is estimated by computing a utility function $U = 1/(1 + r_d)$, where r_d is the remaining distance between the current position of the AV and the target position. In the delayed-reward scheme, no reward is given until the trial ends. When the AV runs out of time, a final reward is computed by using the utility function based on the remaining distance from the AV to the target.

C. Experimental Results

All AVs are based on the TD-FALCON with bounded Q-learning and use the same set of parameter values: choice parameters $\alpha^{ck} = 0.001$ and learning-rate parameters $\beta^{ck} = 1.0$ for $k = 1, 2, 3$; contribution parameters $\gamma^{c1} = \gamma^{c2} = 0.5$, $\gamma^{c3} = 0.0$; and baseline-vigilance parameters $\bar{\rho}^{c1} = 0.5$, $\bar{\rho}^{c2} = 0.2$, $\bar{\rho}^{c3} = 0.5$. For TD learning, the learning rate α is fixed at 0.5, the discount factor γ is set to 0.95, and the initial Q

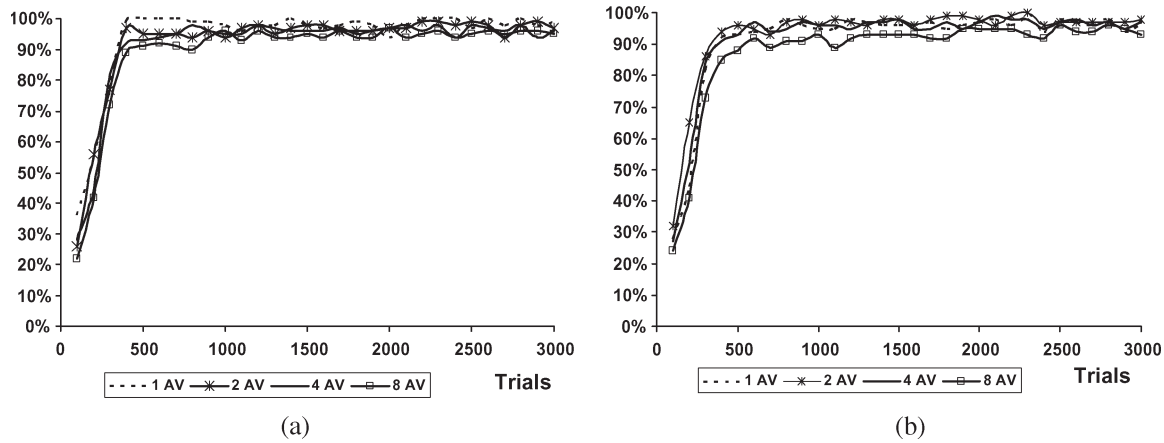

Fig. 3. Success rates of the TD-FALCON teams averaged at 100-trial intervals on the minefield navigation task. (a) Immediate reward. (b) Delayed reward.

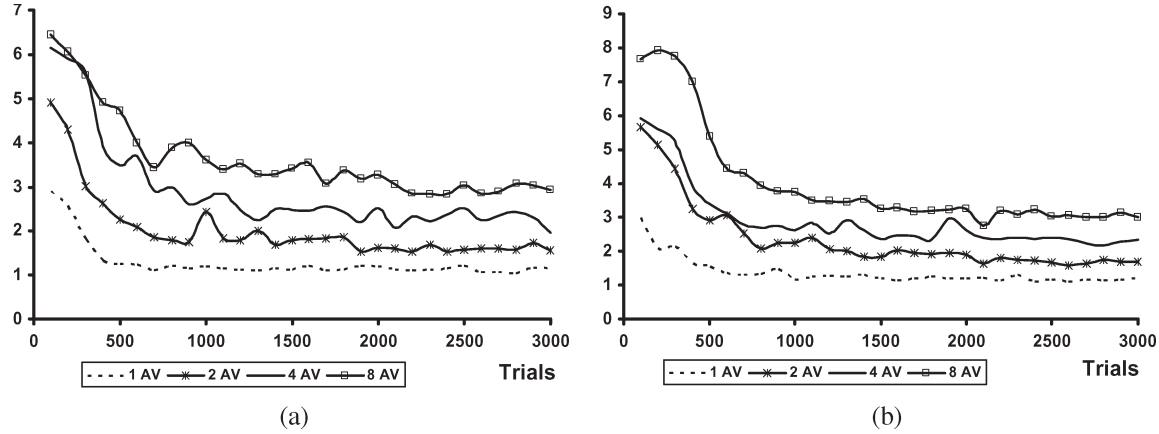

Fig. 4. Normalized steps of the TD-FALCON teams averaged at 100-trial intervals on the minefield navigation task. (a) Immediate reward. (b) Delayed reward.

values are set to 0. For action selection, the decayed ϵ-greedy policy is used with ϵ initialized to 0.6 and decayed at a rate of 0.002.

Each AV learns from scratch based on the feedback signals received from the environment. We run the experiments for 3000 trials. In each trial, the initial locations of the AVs, the location of the target, and the locations of the mines are randomly set. The success rate of a team in a trial is determined by the percentage of the AVs that reach the target position within the specified time in the trial.

Fig. 3 illustrates the performance of the TD-FALCON agent teams consisting of one, two, four, and eight AVs in terms of success rates averaged at 100-trial intervals within 3000 trials. We can see that the success rates of TD-FALCON teams increase rapidly right from the beginning. By the end of 500 trials, almost all teams can achieve more than 90% success rates. Nevertheless, teams with fewer agents produce slightly better results due to the lower chance of collision. In addition, teams with immediate rewards work better than those with delayed rewards. Given the same number of agents, the success rates with immediate rewards are on average of $3\% \sim 5\%$ higher than those with delayed rewards.

To quantitatively evaluate how well the AVs travel from the starting positions to the targets, we define a measure called normalized step, which is given by $\text{step}_n = \text{step}/s_d$, where step is the number of sense–act–learn cycles taken to reach the target and s_d is the shortest distance between the starting and target positions. A normalized step of one means that the AV has taken the optimal (shortest) path to the target.

Fig. 4 depicts the normalized steps taken by the TD-FALCON agent teams consisting of one, two, four, and eight AVs averaged at 100-trial intervals. With just one agent, the normalized step converges to almost one after 1000 trials. Note that the normalized step values can seldom be ones, as detours are unavoidable when mines are on the optimal paths. In general, with more AVs in the system, the paths are always less than optimal as the agents need to avoid collision with the mines, as well as with each other. However, teams with immediate rewards work slightly better than their counterparts with delayed rewards. After 3000 trials, the four-AV team with immediate rewards achieves a normalized step of 1.95, whereas the four-AV team with delayed rewards takes a normalized step of 2.34.

Fig. 5 illustrates the average number of cognitive nodes learned by each agent in the TD-FALCON agent teams consisting of one, two, four, and eight AVs averaged at 100-trial intervals. We observe that the numbers of nodes in all agent teams increase linearly over the learning trials. However, teams with more AVs tend to create a larger number of cognitive nodes, due to the more complex and dynamic environment. Generally, teams with immediate rewards learn a much smaller number of cognitive nodes. After 3000 trials, teams with immediate

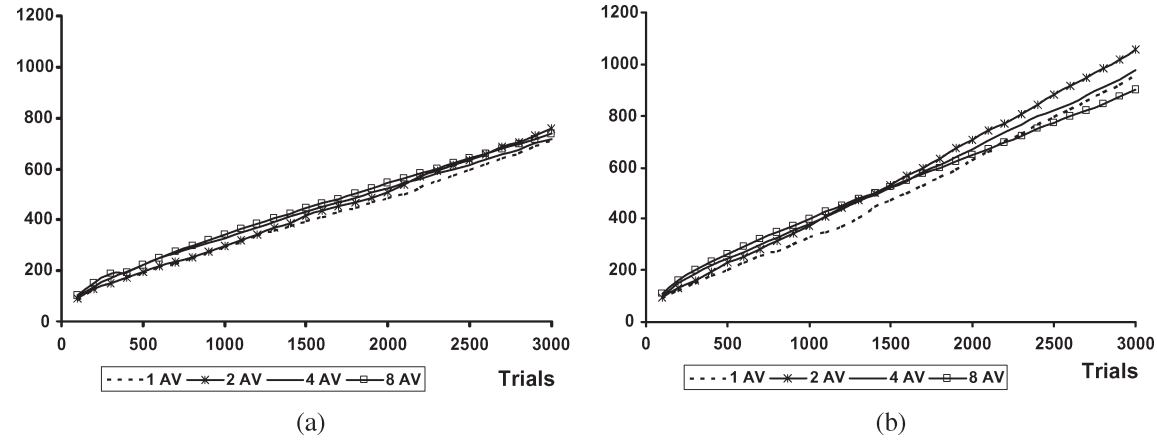

Fig. 5. Number of cognitive nodes created by the TD-FALCON teams averaged at 100-trial intervals on the minefield navigation task. (a) Immediate reward. (b) Delayed reward.

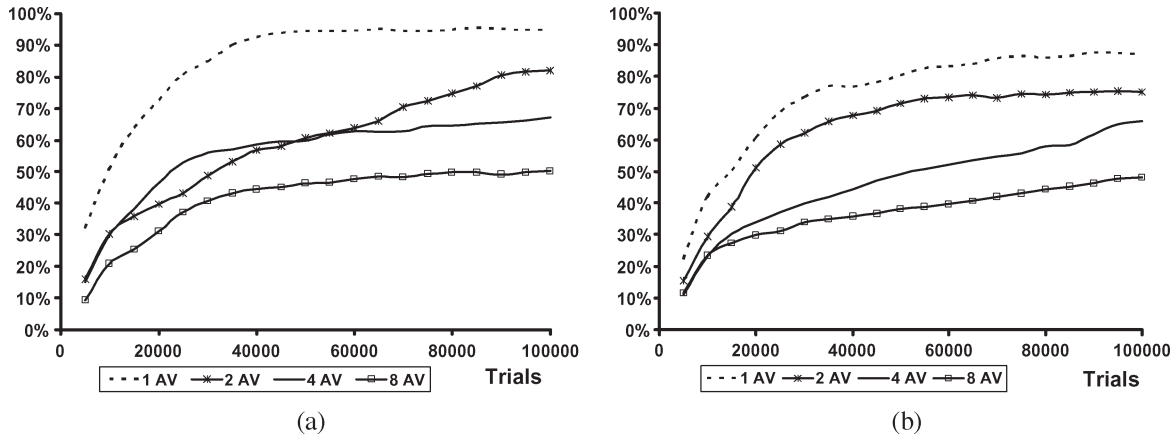

Fig. 6. Success rates of the BP teams averaged at 5000-trial intervals on the minefield navigation task. (a) Immediate reward. (b) Delayed reward.

rewards generate about 700 nodes only, whereas teams with delayed rewards generate about 1000 nodes.

D. Comparing With BP-Based Reinforcement Learners

To put the performance of the TD-FALCON in perspective, we further conduct experiments to evaluate the performance of a traditional RL system, using the standard Q-learning rule and a multilayer feedforward neural network trained with the gradient-descent BP algorithm as the function approximator. We have chosen the BP algorithm as the benchmark of comparison, as it is by far one of the most commonly used function approximators and has been successfully applied to a similar navigation problem [7]. To enable a fair comparison, the BP learner also makes use of the same action selection module based on the decayed ϵ-greedy policy.

The BP learner employs a standard three-layer perceptron architecture to learn the value function with a learning rate of 0.25 and a momentum term of 0.5. The input layer consists of 18 nodes representing the five sonar-signal values, eight possible target bearings, and five selectable actions. The output layer consists of only one node representing the value of performing an action in a particular state. A key issue in using a BP network is the determination of the number of hidden nodes. We experiment with a varying number of nodes empirically and obtain the best results with 36 nodes.

Referring to Fig. 6, we can see that the TD-FALCON evidently outperforms the BP learner in terms of success rates. For example, under the immediate-reward scheme, the four-AV BP teams achieves a success rate of around 67% after 100 000 trials, while the success rate of the four-AV TD-FALCON team after only 3000 trials is 98%. A similar set of performance figures is observed for experiments using the delayed-reward scheme. It is also noticeable that the success rates of the BP teams significantly decrease from 95% with a single agent to 50% with eight agents. In contrast, the TD-FALCON teams can adapt well and maintain the high success rates as the number of agents increases.

In terms of learning speed, the TD-FALCON agent teams also learn much faster than the BP reinforcement learners. For example, under the immediate-reward scheme, the four-AV BP team takes more than 75 000 trials to achieve their peak performance. In contrast, the four-AV TD-FALCON team achieves its peak within just 500 trials. For the experiments with delayed rewards, the same performance difference is also observed. This indicates that TD-FALCON is more than 100 times (over two orders of magnitude) faster than the BP learner in terms of learning efficiency.

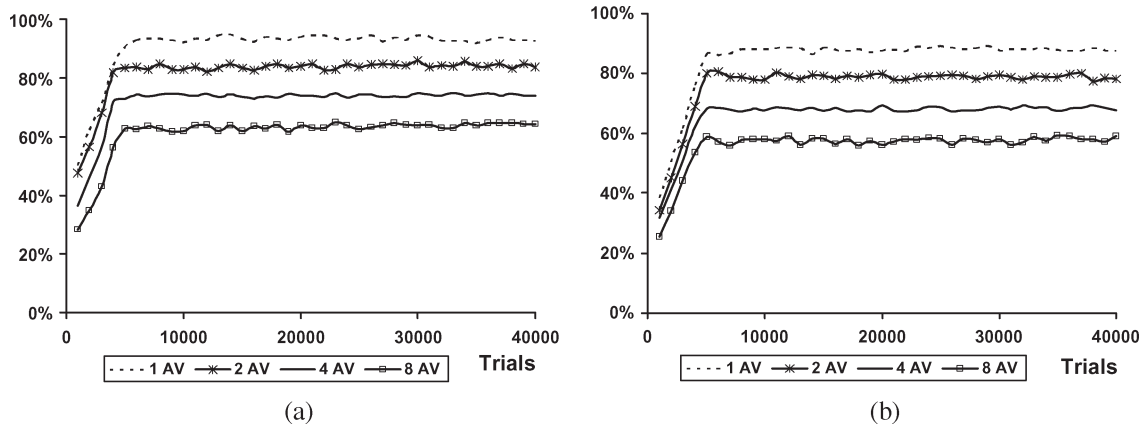

Fig. 7. Success rates of the RPROP teams averaged at 1000-trial intervals on the minefield navigation task. (a) Immediate reward. (b) Delayed reward.

E. Comparing With RPROP-Based Reinforcement Learners

To further validate the performance of TD-FALCON, we repeat our experiments of the gradient-descent-based reinforcement learners, by replacing the standard BP algorithm with the RPROP algorithm [9], [44]. RPROP updates the weights by using the signs of two succeeding derivatives instead of the magnitudes of the derivatives. Compared with standard BP, RPROP is believed to have the advantages of faster learning, lower computation cost, higher robustness in parameter choice, and faster convergence rate.

Similar to the BP learner, the RPROP learner also uses the same action selection policy and a three-layer Perceptron architecture to learn the value function with a learning rate of 0.25. We conduct empirical experiments and obtain the best performance of RPROP teams with 21 hidden units.

As shown in Fig. 7, although RPROP shows a marked improvement over the BP algorithm, its success rates are still significantly lower than those of the TD-FALCON. For example, under the immediate-reward scheme, the four-AV BP teams obtain 67% success rate after 100 000 trials, whereas the four-AV RPROP team can achieve 74.2% success rate after 40 000 trials. However, the success rate of the four-AV TD-FALCON team is 98% after only 3000 trials. Similar to the BP teams, the success rates of the RPROP teams also significantly reduce from 92.8% with one single agent to 64.3% with eight agents under the immediate-reward scheme. In contrast, the TD-FALCON can adapt well and maintain the high success rates in a multiagent environments.

In terms of learning speed, RPROP indeed learns much faster than the BP learners. However, it still does not match the convergence speed of the TD-FALCON. For example, under the immediate-reward scheme, the four-AV RPROP team achieves its peak performance within 5000 trials. However, the four-AV TD-FALCON team achieves peak success rates within only 500 trials. A similar set of performance figures is observed for experiments using the delayed-reward scheme. This indicates that, in terms of learning efficiency, the TD-FALCON is more than 100 times (two orders of magnitude) and ten times (one order of magnitude) faster than the BP learner and RPROP learner, respectively.

To make a direct comparison, Figs. 8 and 9 collate the success rates of BP, RPROP, and TD-FALCON with one and eight agents, respectively, operating under the same conditions for the first 3000 trials.

Referring to Fig. 8, under the immediate-reward scheme, the TD-FALCON learner has a much higher success rate of well above 90% after 3000 trials, compared with 17.6% of BP and 71.4% of RPROP. In addition, the TD-FALCON learner has also a significantly lower normalized step of 1.158, compared with those of the BP and RPROP learners. Experiments using the delayed-reward scheme show similar results.

Referring to Fig. 9, with immediate rewards, the eight-AV TD-FALCON team has a much higher success rate (95.0%) after 3000 trials, compared with 5.7% of the BP team and 43.0% of the RPROP team. On the average, the TD-FALCON team has a normalized step of 2.937, significantly lower than those of both the BP and RPROP teams. Experiments using the delayed-reward scheme, again, show similar results.

F. Scaling up the Navigation Task

To demonstrate the scalability of the TD-FALCON algorithm, we further conduct experiments of the TD-FALCON using a variety of minefield configurations listed as follows:

1) 16 $\times$ 16 minefield with ten mines (original configuration);
2) 32 $\times$ 32 minefield with 20 mines;
3) 48 $\times$ 48 minefield with 30 mines;
4) 64 $\times$ 64 minefield with 40 mines.

Fig. 10 summarizes the success rates of the TD-FALCON teams with four agents in the various minefield configurations. We see that the TD-FALCON teams maintain a high success rate despite the increase in the domain size. With immediate rewards, the success rates after 3000 trials are 98% and 95% in the smallest and largest fields, respectively. Similar results are observed for the delayed-reward scheme.

Nevertheless, we note that the learning speed of TD-FALCON teams gets slower as the minefield size increases. With immediate rewards, the TD-FALCON teams with four agents achieve the peak performance at 500 trials in

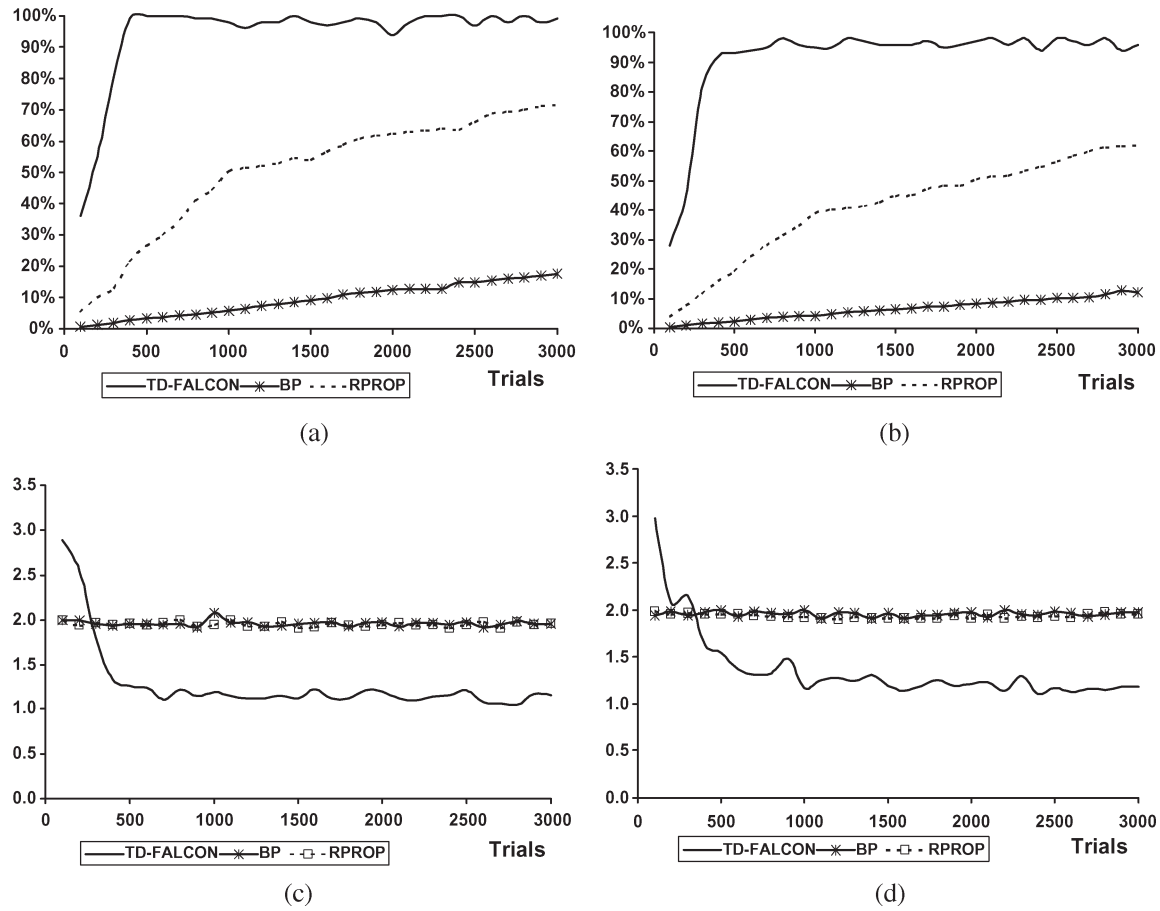

Fig. 8. Performance of the TD-FALCON, BP, and RPROP teams with a single agent averaged at 100-trial intervals on the minefield navigation task. (a) Success rates with immediate rewards. (b) Success rates with delayed rewards. (c) Normalized steps with immediate rewards. (d) Normalized steps with delayed rewards.

the smallest minefield but take up to 1000 trials in the largest one. The same observation is made for the delayed-reward scheme. The increase in learning time is attributed to the longer time needed to explore in a larger minefield.

VI. Predator/Prey Pursuit Game

The predator/prey pursuit domain [10], [45] has been widely used as a benchmark for multiagent systems. Many variants of the pursuit domain exist with varied levels of complexity [46]. To create a challenging problem, our experiments employ eight predators to capture a prey in a 16×16 field. The prey typically starts at the center of the field, and the predators are distributed at the edges of the field (Fig. 11). The rules of the game are summarized as follows.

1) At each time step, each of the predators and the prey can select any of the nine actions: to move in one of the eight possible directions at the pace of one square or to remain stationary. No predator or prey can go out of the game field.
2) The predators attempt to encircle the prey and make it unmovable. When the prey is completely surrounded by the predators, the game ends successfully.
3) The prey moves at the same speed as the predators by adopting the effective maximizing-distance escape strategy [47], by which it selects an action that maximizes the total distance from the predators.
4) When two predators collide or the prey reaches any side of the game field, the game is deemed to have failed. When a pursuit is not completed within 30 steps, it is also considered as a failure.

A. State Representation

For the purpose of cooperation, an agent needs to be aware of its surrounding agents. Using monolithic Q-learning, the state space of each agent is increased with the information of the other agents [34], [35], [48]. The growth of the dimensionality of the state space for each agent is exponential with respect to the number of agents, leading to the problem of combinatorial explosion [49]. As an illustration, consider a world with n predators and one prey, the state of a predator i can be represented by an n-tuple $(c_1, c_2, \ldots, c_n)$, where c_i represents the relative positions of the prey to the predator i and $c_j (j \neq i)$ indicates the relative positions of the predator j with respect to the predator i. For a pair of predators (or a predator and a prey), the possible cases of relative positions between them could be $(2d+1)^2$, where d is the visual depth of a predator. In our game field, $d = 15$, so the number

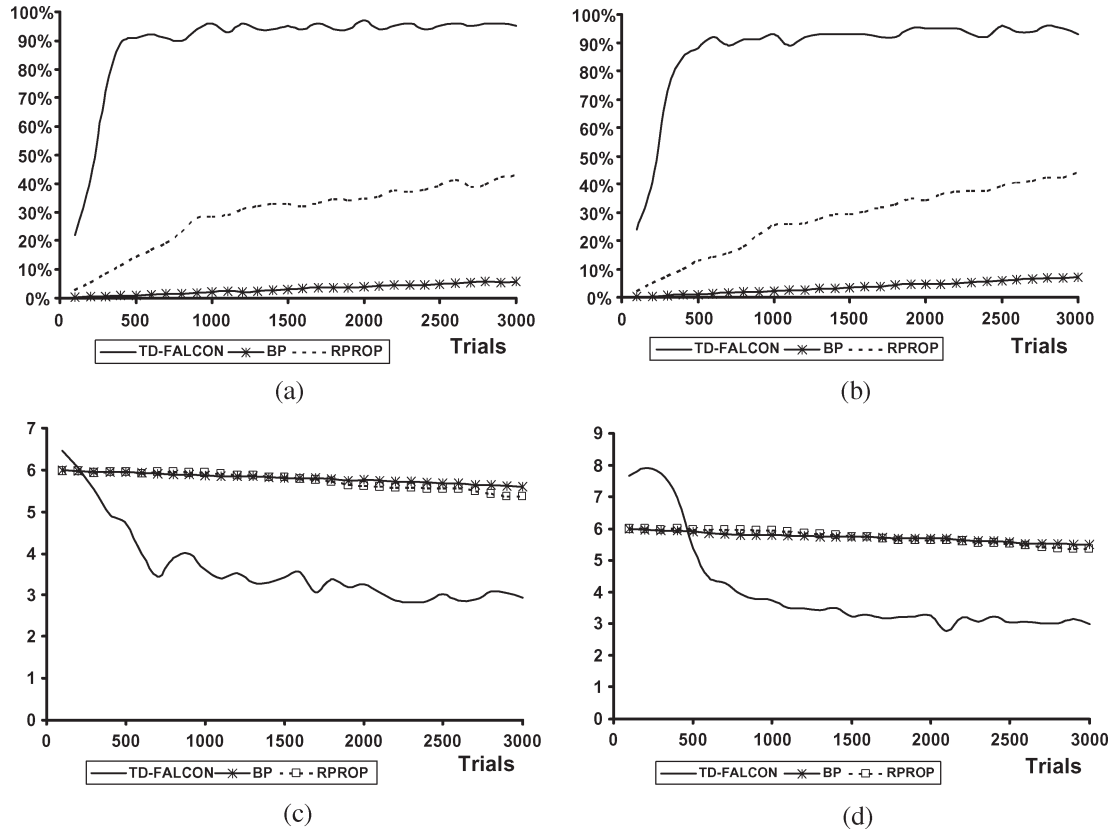

Fig. 9. Performance of the TD-FALCON, BP, and RPROP teams with eight agents averaged at 100-trial intervals on the minefield navigation task. (a) Success rates with immediate rewards. (b) Success rates with delayed rewards. (c) Normalized steps with immediate rewards. (d) Normalized steps with delayed rewards.

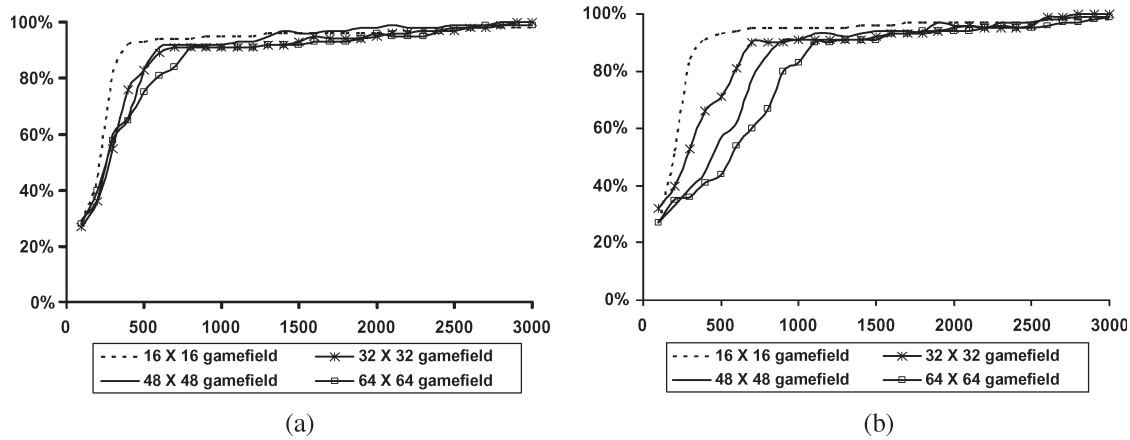

Fig. 10. Success rates of the TD-FALCON teams consisting of four agents averaged at 100-trial intervals under various minefield configurations. (a) Immediate reward. (b) Delayed reward.

of possible states that a predator can have could be $N = (2d+1)^2 = (2 \times 15 + 1)^2 = 961$. Since there are eight sets of coordinates to track, the size of the state space for a single predator balloons to $S = N^8 = 961^8 = 727,423,121,747,185,263,828,481$.

1) All-Bearing Strategy: Similar to the minefield navigation problem, our formulation of the predator/prey pursuit game also follows the POMDP. Based on our analysis of the problem, the optimal action of a predator depends on its bearing rather than on its distance to the prey. This implies that the canonical distance between the predators and the prey may not be strictly required. Suppose that the bearing from the predator i to the prey is b^i, where $0 \leq b^i \leq 7$, the prey bearing vector $\mathbf{B}^i$ is computed by

$$B^i_j = \begin{cases} 1, & \text{if } j = b^i \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

and the other bearing vector $\mathbf{B}^o$ is the concatenation of the bearing vectors from the other predators to the prey. Considering bearing values for just eight directions, the combined state vector translates to an 8×8 state space.

2) Center of Agent Team (CAT) Bearing Strategy: Instead of representing the bearing of each agent, we introduce a

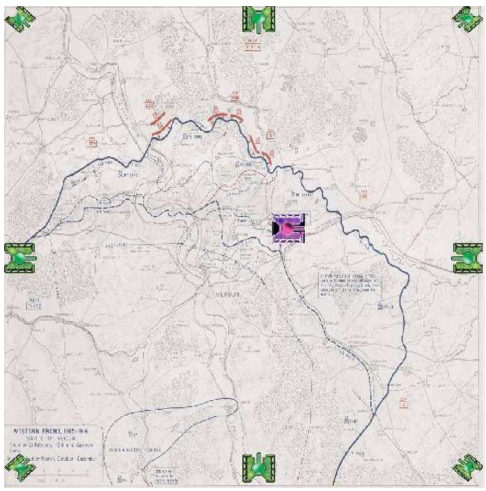

Fig. 11. Layout of the pursuit game.

TABLE II
STATE VECTORS AND REWARD FUNCTIONS USED BY THE VARIOUS COOPERATIVE STRATEGIES

Strategy	State Representation	Reward (r^i)
Non-cooperating	$\mathbf{D} = (s_0, s_1)$ $\mathbf{S^i} = (a_0, a_1, \cdots, a_{15})$ $\mathbf{B^i} = (b_0, b_1, \cdots, b_7)$	$\frac{1}{(1+d^i)}$ 0 if fail
All Bearing	$\mathbf{S^i} = (a_0, a_1, \cdots, a_{15})$ $\mathbf{B^i} = (b_0, b_1, \cdots, b_7)$ $\mathbf{B^o} = (b_8, b_9, \cdots, b_{31})$	$\frac{1}{(1+d^i)(1+d^c)}$ 0 if fail
CAT Bearing	$\mathbf{S^i} = (a_0, a_1, \cdots, a_{15})$ $\mathbf{B^i} = (b_0, b_1, \cdots, b_7)$ $\mathbf{B^c} = (c_0, c_1, \cdots, c_8)$	$\frac{1}{(1+d^i)(1+d^c)}$ 0 if fail

high-level concept called CAT and incorporate into the state vector the bearing from the CAT to the prey. Whereas a typical predator bearing vector consists of eight possible values, the CAT bearing vector has an extra value denoting the situation that the CAT coincides with the prey. Suppose that the bearing from the CAT to the prey is b^c (where $0 \leq b^c \leq 8$), the CAT bearing vector $\mathbf{B}^c$ is computed by

$$B_j^c = \begin{cases} 1, & \text{if } j = b^c \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

To avoid colliding with other predators, each predator is equipped with an additional set of eight sonar inputs, one for each direction. With complement coding, there is a total of 16 sonar values. To simplify the problem, we adopt binary sonar inputs to detect the existence of predators in the immediate surrounding. The predator sonar vector $\mathbf{S}^i = (a_0, a_1, \ldots, a_{15})$ is given by

$$a_j = \begin{cases} 1, & \text{if } j < 8 \text{ and } d_j = 1 \\ 0, & \text{if } j < 8 \text{ and } d_j > 1 \\ 1 - a_{j-8}, & \text{if } j \geq 8 \end{cases} \tag{13}$$

where $d_j (0 \leq j \leq 7)$ is the distance between the predator and another predator in the jth bearing.

Using the CAT bearing strategy, the complete state vector of the predator i is thus given by $(\mathbf{S}^i, \mathbf{B}^i, \mathbf{B}^c)$. In total, the size of the state space for each predator is $2^8 \times 8 \times 9 = 18\,432$, much less than the solution based on relative positions.

B. Reward Scheme

In a canonical TD-FALCON algorithm, an agent is not aware of how its movement contributes to the entire team and the outcome of the game. Previously, profit sharing [50] has been used by averaging the rewards received by all the agents. However, this approach fails to consider the specific situation of each individual agent. In a successful trial, the prey is eventually immovable after being surrounded by the eight predators. When this happens, the distance between each predator and the prey is the minimum, and the CAT coincides with the prey. When a predator is pursuing the prey, the distance between them is decreasing, and the CAT is also approaching the prey. Therefore, a predator makes a contribution to the overall task by a move that reduces its distance to the prey and, at the same time, gets the center closer to the prey. To incorporate both the individual and team payoffs, the reward function of a predator i is defined as $r_i = 1/(1 + d^i)(1 + d^c)$, where d^i and d^c are the distances from the predator and the CAT to the prey, respectively.

When the predators successfully surround the prey, the above reward function produces a value of one. However, when two predators collide, a final reward signal of zero is given. Likewise, when the prey reaches the edge of the game field, a reward of zero is provided to all the agents.

C. Experimental Results

We conduct three sets of comparative experiments to evaluate the efficacies of the various cooperative strategies. The first set of the experiments, involving teams of noncooperating agents with relative positional information in the state representation, provides the baseline performance for comparison. The relative positional information is denoted as $\mathbf{D} = (s_0, s_1)$, where $\mathbf{s_0} = 1/(1 + d)$, $s_1 = 1 - s_0$ and d is the distance from this agent to the prey. The second set of the experiments involves the TD-FALCON teams using the all-bearing cooperative strategy. The last set of the experiments employs the TD-FALCON teams using the CAT-bearing cooperative strategy. The sensory state representations and the reward functions used in the various experiments are summarized in Table II. All experiments use the following parameter values: choice parameters $\alpha^{ck} = 0.001$, learning rates $\beta^{ck} = 1.0$ (fast learning), baseline-vigilance parameters $\bar{\rho}^{ck} = 0.9$ for $k = 1$ to 3, initial exploration rate $\epsilon = 0.6$, decaying rate $d_\epsilon = 0.0004$, Q-learning-rule learning rate $\alpha = 0.5$, and discount parameter $\gamma = 0.01$.

Fig. 12 shows that the cooperating TD-FALCON teams have a clear performance advantage over their noncooperating counterparts. After 6500 trials, the success rates of cooperating teams reach 95%, compared with less than 85% success rates achieved from noncooperating teams.

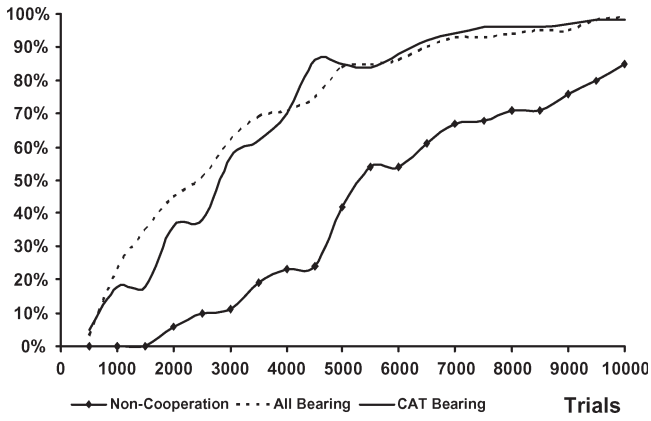

Fig. 12. Success rates of the TD-FALCON teams using the three strategies on the pursuit game.

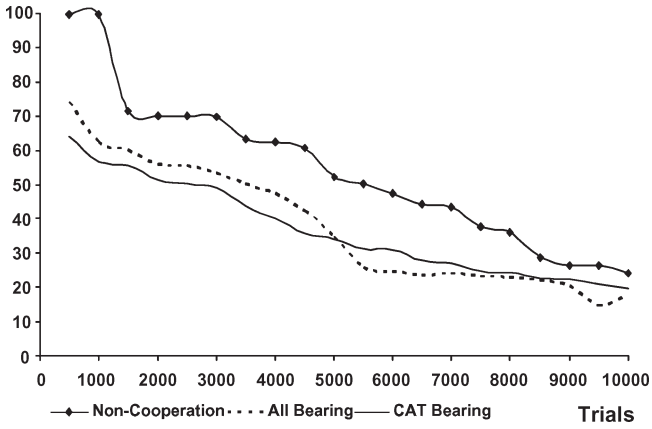

Fig. 13. Number of steps taken by the TD-FALCON teams using the three strategies on the pursuit domain.

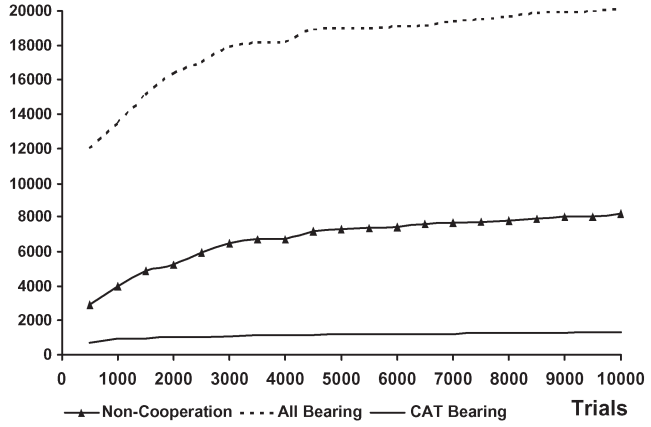

Fig. 14. Number of cognitive nodes learned by the TD-FALCON teams using the three strategies on the pursuit domain.

Fig. 13 again illustrates the benefit of cooperation in terms of the average number of steps taken by a predator team to surround the prey. In spite of slight fluctuations, the trend clearly shows that the cooperating teams are more efficient than the noncooperating teams.

Fig. 14 compares the three strategies based on the number of cognitive nodes created by each individual TD-FALCON agent. We observe that the use of CAT bearing results in a significantly smaller number of cognitive nodes learned. The significant code reduction is believed to be the result of the removal of the relative positional information, as well as the bearings of other agents, from the state vector.

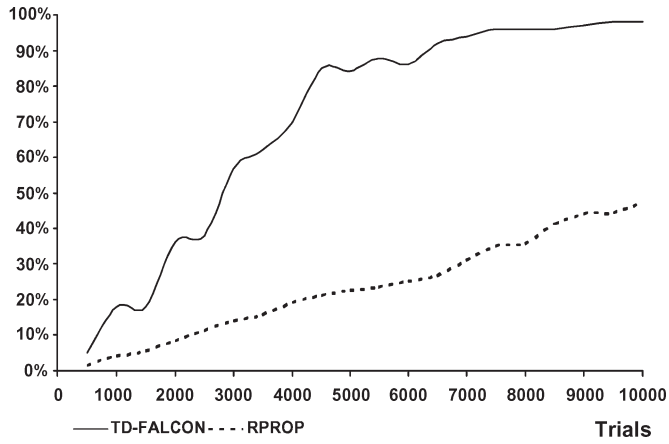

Fig. 15. Success rates of the TD-FALCON and RPROP teams using the CAT bearing strategy on the pursuit domain.

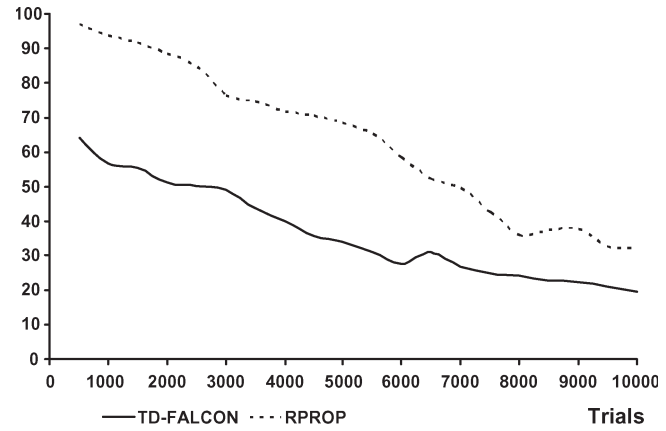

Fig. 16. Number of steps taken by the TD-FALCON and RPROP teams using the CAT bearing strategy on the pursuit domain.

D. Comparing With RPROP

To put the performance of the TD-FALCON in perspective, we repeat the experiments using the same setting, with the RPROP reinforcement learners. Fig. 15 shows the success rates obtained by the RPROP team over 10 000 trials averaged over ten runs of experiments. We see that the RPROP team generally takes a longer time to learn and yet achieves a significantly lower success rate than that of the TD-FALCON. After 10 000 trials, the success rate of the TD-FALCON team is typically above 95%. However, the RPROP team only achieves less than 50% success rate.

Fig. 16 shows the number of steps taken by the RPROP team to capture the prey within 10 000 trials, averaged over ten runs of experiments. It can be noticed that the RPROP team takes more steps to capture the prey than the TD-FALCON team. After 10 000 trials, the TD-FALCON team takes only 19.50 steps to capture the prey on average, while the RPROP team requires an average of 32.57 steps to fulfill the task.

VII. Conclusion

Two critical issues in multiagent cooperative learning systems are scalability and dynamic coadaptation. By using a self-organizing learning architecture, the TD-FALCON offers an efficient solution to the scalability problem when dealing with large and continuous state–action spaces. Specifically, its online incremental learning capability enables an agent to adapt and to operate in a real-time multiagent environment.

In our minefield experiments, we show that the TD-FALCON systems can produce superior performance, in terms of success rate, stability, and learning efficiency, in the presence of other learning agents. Our experiments on the multiagent predators/prey pursuit task indicate that appropriate coding of state representation can greatly enhance the scalability of the multiagent systems. In addition, high-level cooperative signals, such as CAT, are effective in generating outstanding performance with high efficiency and scalability.

The TD-FALCON can be seen as a function-approximation method for temporal-difference learning. Although it has been widely acknowledged that the convergence of Q-learning is longer guaranteed with the use of function approximators, the TD-FALCON has demonstrated superior performance in terms of success rates and stability across our empirical experiments.

On the other hand, although the TD-FALCON has shown encouraging results, the work reported here only marks the beginning of our study into the multiagent domain. The cooperative tasks that we have studied so far are standard benchmark tasks. While they serve the purpose of evaluating and comparing the performance of the TD-FALCON agents with alternative methods empirically, it remains a challenge to maintain the robustness and adaptability of the TD-FALCON in more complex real-world problems. Our future work will include the development of more generic cooperative strategies that can cope with a variety of complex and dynamic situations.

Acknowledgment

The authors would like to thank the three anonymous reviewers for providing many valuable comments and suggestions to the previous version of this paper.

References

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

[2] W. S. Lovejoy, "Computationally feasible bounds for partially observed Markov decision processes," *Oper. Res.*, vol. 39, no. 1, pp. 162–175, Jan./Feb. 1991.

[3] A. H. Tan, "FALCON: A fusion architecture for learning, cognition, and navigation," in *Proc. IJCNN*, Budapest, Hungary, 2004, pp. 3297–3302.

[4] A.-H. Tan and D. Xiao, "Self-organizing cognitive agents and reinforcement learning in a multi-agent environment," in *Proc. IEEE/ACM/WIC Int. Conf. Intell. Agent Technol.*, 2005, pp. 351–357.

[5] A. H. Tan, N. Lu, and D. Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Trans. Neural Netw*. to be published.

[6] D. Gordan and D. Subramanian, "A cognitive model of learning to navigate," in *Proc. 19th Annu. Conf. Cognitive Sci. Soc.*, 1997, pp. 271–276.

[7] R. Sun, E. Merrill, and T. Peterson, "From implicit skills to explicit knowledge: A bottom-up model of skill learning," *Cogn. Sci.*, vol. 25, no. 2, pp. 203–244, Mar. 2001.

[8] M. Riedmiller and H. Braun, "RPROP—A fast adaptive learning algorithm," Univ. Karlsruhe, Karlsruhe, Germany, 1992. Tech. Rep. (Also Proc. of ISCIS VII).

[9] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Netw.*, San Francisco, CA, 1993, pp. 586–591.

[10] M. Brenda, V. Jagannathan, and R. Dodhiawala, "On optimal cooperation of knowledge sources—An empirical investigation," Boeing Adv. Technol. Center, Boeing Comput. Services, Seattle, WA, Tech. Rep. BCS-G2010-28, 1986.

[11] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," George Mason Univ., Fairfax, VA, Tech. Rep. GMU-CS-TR-2003-1, 2003.

[12] M. Dubreuil, C. Gagne, and M. Parizeau, "Analysis of a master–slave architecture for distributed evolutionary computations," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 229–235, Feb. 2006.

[13] H. P. Schwefel, *Advantages and Disadvantages of Evolutionary Computation Over Other Approaches, Evolutionary Computation 1: Basic Algorithms and Operators*. Bristol, PA: Inst. Phys. Publishing, 2000.

[14] M. Kaya and R. Alhajj, "Modular fuzzy-reinforcement learning approach with internal model capabilities for multiagent systems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 1210–1223, Apr. 2004.

[15] M. Kaya and R. Alhajj, "Fuzzy OLAP association rules mining-based modular reinforcement learning approach for multiagent systems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 326–338, Apr. 2005.

[16] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.

[17] H. Kose, U. Tatlidede, C. Mericli, K. Kaplan, and H. L. Akin, "Q-learning based market-driven multi-agent collaboration in robot soccer," in *Proc. Turkish Symp. Artif. Intell. Neural Netw.*, Izmir, Turkey, 2004, pp. 219–228.

[18] V. Kononen, "Gradient descent for symmetric and asymmetric multiagent reinforcement learning," *Web Intell. Agent Syst.: Int. J. (WIAS)*, vol. 3, no. 1, pp. 17–30, 2005.

[19] E. F. Yang and D. B. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," Dep. Comput. Sci., Univ. Essex, Colchester, U.K., Tech. Rep. CSM-404, 2004.

[20] K. S. Hwang, S. W. Tan, M. C. Hsiao, and C. S. Wu, "Cooperative multiagent congestion control for high-speed networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 255–268, Apr. 2005.

[21] L. Bull and T. C. Fogarty, "Evolving cooperative communicating classifier systems," in *Proc. 4th Annu. Conf. Evol. Program.*, 1994, pp. 308–315.

[22] T. Balch, "Reward and diversity in multirobot foraging," in *Proc. Agents Learning About, From With Other Agents Workshop*, Stockholm, Sweden, 1999.

[23] J. M. Vidal and E. H. Durfee, "The moving target function problem in multi-agent learning," in *Proc. 3rd Int. Conf. Multi-Agent Syst.*, Paris, France, Jul. 1998, pp. 317–324.

[24] G. Chalkiadakis and C. Boutilier, "Coordination in multiagent reinforcement learning: A Bayesian approach," in *Proc. 2nd Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2003, pp. 709–716.

[25] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Adv. Complex Systems*, vol. 4, no. 2/3, pp. 265–279, 2001.

[26] T. Balch, "Learning roles: Behavioural diversity in robot teams," Georgia Inst. Technol., Atlanta, GA, Tech. Rep. GIT-CC-97-12, 1997.

[27] M. Mataric, "Learning to behave socially," in *Proc. 3rd Int. Conf. Simul. Adaptive Behaviour*, 1994, pp. 453–462.

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

[29] M. Bowling and M. Veloso, "An analysis of stochastic game theory for multiagent reinforcement learning," Comput. Sci. Dept., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-00-165, 2000.

[30] R. Mukherjee and S. Sen, "Towards a Pareto-optimal solution in general-sum games," in *Proc. 2nd Int. Joint Conf. AAMAS*, Melbourne, Australia, 2003, pp. 153–160.

[31] M. Mundhe and S. Sen, "Evaluating concurrent reinforcement learners," in *Proc. 4th ICMAS*, Boston, MA, 2000, pp. 421–422.

[32] S. Sen and M. Sekaran, "Individual learning of coordination knowledge," *J. Exp. Theor. Artif. Intell.*, vol. 10, no. 3, pp. 333–356, Jul. 1998.

[33] S. Sen, M. Sekaran, and J. Hale, "Learning to coordinate without sharing information," in *Proc. 12th Nat. Conf. Artif. Intell.*, Seattle, WA, 1994, pp. 426–431.

[34] M. L. Littman, "Markov games as a framework for multiagent reinforcement learning," in *Proc. 11th Int. Conf. Machine Learning*, San Francisco, CA, 1994, pp. 157–163.
[35] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Machine Learning*, 1993, pp. 330–337.
[36] J. Hu and M. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm," in *Proc. 15th Int. Conf. Machine Learning*, 1998, pp. 242–250.
[37] A. Agogino and K. Tumer, "Reinforcement learning in large multi-agent systems," in *Proc. AAMAS Workshop Coordination Large Scale Multi-Agent Syst.*, Utrecht, The Netherlands, 2005.
[38] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vis. Graph. Image Process.*, vol. 37, no. 1, pp. 54–115, Jan. 1987.
[39] G. A. Carpenter and S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog input patterns," *Appl. Opt.*, vol. 26, no. 23, pp. 4919–4930, Dec. 1987.
[40] B. Moore, "ART 1 and pattern clustering," in *Proc. Connectionist Models Summer School*, 1988, pp. 174–185.
[41] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Netw.*, vol. 4, pp. 759–771, 1991.
[42] A. Pérez-Uribe, "Structure-adaptable digital neural networks," Ph.D. dissertation, Swiss Federal Inst. Technol.–Lausanne, Lausanne, Switzerland, 2002.
[43] R. Sun, T. Peterson, and E. Merrill, "Bottom-up skill learning in reactive sequential decision tasks," in *Proc. 18th Cognitive Sci. Soc. Conf.*, 1996, pp. 684–690.
[44] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "A new learning rates adaptation strategy for the resilient propagation algorithm," in *Proc. ESANN*, 2004, pp. 1–6.
[45] P. Stone, "Layered learning in multiagent systems," in *Proc. AAAI/IAAI*, 1997, p. 819.
[46] J. Denzinger and M. Fuchs, "Experiments in learning prototypical situations for variants of the pursuit game," in *Proc. 2nd ICMAS*, Kyoto, Japan, 1996, pp. 48–55.
[47] J. Denzinger and A. Schur, "On customizing evolutionary learning of agent behavior," in *Proc. Can. Conf. AI*, 2004, pp. 146–160.
[48] T. W. Sandholm and R. H. Crites, "Multiagent reinforcement leaning in the iterated prisoner's dilemma," *Biosystems*, vol. 37, no. 1, pp. 147–166, 1995.
[49] N. Ono and K. Fukomoto, "Multi-agent reinforcement learning: A modular approach," in *Proc. 2nd Int. Conf. Multi-Agent Syst.*, 1996, pp. 252–258.
[50] S. Arai and K. Sycara, "Effective learning approach for planning and scheduling in multi-agent domain," in *Proc. 6th Int. Conf. Simulation Adaptive Behaviour (From Animals to Animats 6)*, 2000, pp. 507–516.

Dan Xiao received the B.S. degree from Beijing University, Beijing, China, in 1992 and the Master of Applied Science degree from the School of Applied Science, Nanyang Technological University, Singapore, Singapore, in 2000, where he has been working toward the Ph.D. degree in the School of Computer Engineering since 2004.

His research areas include cluster-based systems and multiagent learning.

Ah-Hwee Tan (M'04–SM'04) received the B.Sc. (with first class honors) and M.Sc. degrees in computer science from the National University of Singapore, Singapore, and the Ph.D. degree in cognitive and neural systems from Boston University, Boston, MA.

He is an Associate Professor and the Director with the Emerging Research Laboratory, School of Computer Engineering, Nanyang Technological University, Singapore. He is also a Faculty Associate with the A $*$ STAR Institute for Infocomm Research, Singapore, where he was formally the Manager of the Text Mining and Intelligent Cyber Agents Groups. He is a member of the Editorial Board of *Applied Intelligence*. His current research areas include cognitive and neural systems, intelligent agents, machine learning, media fusion, and information mining. He is the holder of several patents and has successfully commercialized a suite of document analysis and text-mining technologies.

Dr. Tan is a member of Association for Computing Machinery.