Self-Organizing Neural Architectures and Multi-Agent Cooperative Reinforcement Learning

Xiao Dan

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree
of Doctor of Philosophy

2010

Abstract

Multi-agent system, wherein multiple agents work to perform tasks jointly through their interaction, is a fairly well studied problem. Many approaches to multi-agent learning exist, among which, reinforcement learning is widely used, as it does not require an explicit model of the environment. However, limitations remain in current multi-agent reinforcement learning approaches, including adaptability and scalability in complex and specialized multi-agent domains. In any multi-agent reinforcement learning system, two major considerations are the reinforcement learning methods used and the cooperative strategies among agents. In this research work, we propose to adopt a self-organizing neural network model, named Temporal Difference - Fusion Architecture for Learning, COgnition, and Navigation (TD-FALCON), for multi-agent reinforcement learning. TD-FALCON performs online and incremental learning in real-time with and without immediate reward signals. It thus enables an agent to learn effectively in a dynamic environment.

Based on the TD-FALCON model, we develop several cooperative reinforcement learning strategies, investigating how a team of FALCON agents may cooperate to learn and function in a dynamic multi-agent environment. We first experiment TD-FALCON in minefield navigation domain, which does not require an explicit mechanism of collaboration. Our experiments show that TD-FALCON agent teams are able to adapt and function well, comparing with the backpropagation (BP) based as well as the resilient propogation (RPROP) based reinforcement learners. In multi-agent tasks requiring explicit collaboration, scalability becomes an issue as each agent needs to communicate and share information with the other agents. To address the scalability issue, we propose the center of agent team

(CAT) cooperative strategy to reduce the state space, by using a compressed state representation. Our experiments on a predator/prey pursuit task show that the CAT strategy produces the best results in terms of both efficiency and performance, comparing with the non-cooperating strategy and the all-bearing strategy. In more diverse and complex multi-agent environments, wherein an agent may receive more signals from other agents, we present a neighboring-agent mechanism (NAM), wherein an agent receives and processes only sensory inputs from neighboring agents. Experiments in a herding game, wherein multiple shepherds are tasked to encircle and force a sheep to enter a corral, indicate that the NAM strategy enables a TD-FALCON team to remain functional and adaptable.

In addition to the typical multi-agent domains, this thesis further considers a special type of topology-based multi-agent systems (TMAS), wherein agents interact with one another according to their spatial relationship. In a TMAS domain, each agent may have a different state space, therefore traditional approaches to multi-agent cooperative learning may not be able to scale up with the complexity of the network topology. Addressing the issue, we propose a cooperative learning strategy, namely TD-FALCON Binary Tree Formation (BTF), wherein autonomous agents are assembled in a binary tree formation, to effectively unify the state space of individual agents and enable knowledge sharing across agents. Experiments based on a generic network routing (NR) problem show that a TD-FALCON BTF team can perform well under various scales of network complexity and traffic volume.

Keywords: Temporal difference methods, self-organizing neural architectures, reinforcement learning, multi-agent cooperative learning, center of agent team, neighboring-agent mechanism, topology-based multi-agent systems, binary tree formation, knowledge sharing

Acknowledgment

I am really grateful to my supervisor, Dr. Tan Ah Hwee, for his patient guidance and invaluable advices for this thesis and all that I've achieved so far. I would also like to acknowledge the support from my family members, especially my wife, my father and my father-in-law, who stand firmly behind me all the time. This work is also a precious gift for my growing daughter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cooperative works are ubiquitous in our daily life. We can notice that many tasks require the cooperation of multiple autonomous entities to fulfil. Such a task can be an IT project, a curriculum system of a school, or a soccer match. For the case of an IT project, the autonomous entities in cooperation would consist of project managers, system architects, programmers, and test engineers. For the case of a school curriculum system, the autonomous entities refer to the teachers, who cooperate to formulate an effective timetable, avoiding any conflict. For a soccer game (Figure 1.1), an autonomous entity can be the coach or an individual player in a soccer team. The coach can make the strategy and the tactics for the whole team. At any time during the match, each individual player should decide how to perform the ball work, such as passing, shooting, ball control, heading, and so on. In addition, varied players of the soccer team, including forwards, defenders, midfielders and the goal keeper, need to cooperate tightly to fulfil the task of winning the game.

Such cooperative works have been theorized, systemized and abstracted in the research field of distributed artificial intelligence (DAI) [1, 2, 3], wherein an autonomous entity in a cooperative task is defined as an intelligent agent (or agent) [4]. Agents observe and act upon an environment and direct their activities towards achieving goals, by learning and/or using knowledge. They may be very simple or very complex. An agent can be a young

Figure 1.1: A multi-agent domain.

tiger learning preying skills, a human being, or even a community of human beings working together towards a goal. Since a learning process is necessary for multiple cooperative agents to fulfil a task, a subfield of AI, namely multi-agent cooperative learning, has been developed for a number of years [3, 1]. So far there are mainly two categories of learning approaches, namely *evolutionary computation (EC)* and *reinforcement learning (RL)*, in multi-agent cooperative learning [3]. EC is computationally intensive and thus it is not widely used in real-time applications [5]. Our research work in multi-agent cooperative learning is based on reinforcement learning, 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 [6, 7].

1.1 Motivation

Multi-agent systems are computational systems in which two or more agents interact or work together to perform some sets of tasks or to satisfy some sets of goals [1]. Multi-agent problem complexity can rise rapidly with the growth in the number of agents, due to the interactions among the agents. The adaptation task of an autonomous agent in a multi-agent system is challenging, for the following two reasons: first of all, an agent is affected by the actions of the other agents and its action also affects the other agents and the environment [8]; secondly, as agents continue to learn, their behaviors change over time and the environment becomes highly dynamic. So far there have been some great progresses achieved in multi-agent learning [3, 9, 10]. However, the limitations of adaptability and scalability remain in complex and specialized multi-agent systems. The key issues are described in details as follows.

1. **Scalability and stablility of Reinforcement Learning Algorithms.** A popularly used reinforcement learning method is Q-learning [11], which requires an agent to maintain a table to store each and every tuple of state, action, and Q-values. Such a requirement clearly causes a scalability problem. To relieve the scalability problem of Q-learning, many approaches based on function approximation have been proposed [12, 13]. Among the approaches, multi-layer perceptron (MLP) [14, 15, 16] with the gradient descent based backpropagation learning algorithms has been widely used. However, those backpropagation learning algorithms typically require an iterative learning process. Hence the resultant systems may not be able to learn and operate in real time multi-agent systems with a rapidly changing environment. Additionally, the backpropagation learning algorithms may not produce a stable performance in a multi-agent environment, because learning of new patterns may erode the previously learned knowledge.

2. **Multi-agent Scalability.** An agent may need to consider other agents' states and/or actions before taking any action. Hence, the task of an agent in a multi-agent environment becomes more complex. Due to the increased complexity, existing works in

multi-agent learning are usually restricted to a limited number of players. For example, Littman [17] proposes a minimax Q-learning algorithm to update the V values, which associate each pair of state and action to a utility value in reinforcement learning, in two-player zero-sum stochastic games and Tan [18] shows that 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. Hu and Wellman [19] propose the Nash-Q algorithm to update the V values based on some Nash equilibrium. Unfortunately, the solution is also restricted to only two players further with the condition that both agents learn the same Nash equilibrium.

3. **Specialized multi-agent domains.** The last but not the least, the above mentioned limitations escalate in specialized topology-based multi-agent systems (TMAS), wherein agents interact with one another according to their spatial relationship. In the TMAS domains, a straightforward solution, namely *single agent strategy*, has been widely used [20, 21, 22]. For example, using the single agent strategy, a network routing (NR) system, which is a typical TMAS domain, employs one autonomous agent for each routing node, wherein each agent learns an optimal policy through the sensory and reinforcement feedback from its neighbors and the environment. However, the single agent strategy as used in the TMAS domains is facing great challenges. First of all, the agents have to handle the added spatial constraints. With a unique topological relationship with its neighbors, each agent has to deal with a different set of state space. Additionally, as the number of agents increases, the agents have to adapt to the increased complexity of the spatial relationship.

1.2 Objectives

To address the above issues from the existing research work in typical multi-agent domains, we need to develop a learning system that is able to function and adapt well in multi-agent tasks. We also aim to develop effective cooperative strategies in multi-agent

learning environment. The cooperative strategies should be scalable, by reducing the state and/or action space of an agent and enabling agents to learn and act without explicit knowledge of the other agents' states and behaviors. We should also take into account each agent's contribution to the whole team's task when setting up the reward scheme for the cooperative strategy.

In addition, to overcome the limitations of existing works in TMAS domains, we need to provide a strategy to curb the expansion of the state space with the growth of the complexity of the network topology. Our objective is to realize knowledge sharing across all agents in a TMAS domain, thus improving the learning and operational efficiency [18]. For the purpose of performing knowledge sharing, it is desirable to establish a uniform state space representation among all agents in the TMAS domain.

1.3 Approach and Methodology

In our quest towards a functional, adaptable and scalable multi-agent system that is able to work in various multi-agent environments, this dissertation research has followed the design principles listed below.

1. We prefer to adopt *concurrent learning*, under which an agent team employs multiple learners to conduct the learning process simultaneously. In other words, we do not favour *team learning*, whereby an agent team employs a single learner to do the learning process [3]. This choice on the model of cooperative strategies is simply based on the consideration that team learning has the disadvantages of large state space and resource centralization.

2. Agents should be able to learn in an online and incremental manner so as to improve the co-adaptation among agents. In a dynamic multi-agent system, the state of the environment keeps on changing over time due to the interactions among multiple agents [23]. As such, those agents must be able to adapt to each other and the changing environment. Additionally, a multi-agent task typically requires a long and

continuous cooperative relationship. Therefore, those agents involved should be able to maintain stable knowledge and performance during the learning process.

3. Agents should be able to learn in an environment with immediate and/or delayed reward. In real-world multi-agent environments, it is often difficult for an agent to get immediate reward, because the signals from the target may be blocked or become invisible during the learning process [24].

4. It is preferred that *0-level* modeling agents [3] should be used for teammate modeling. Generally, a 0-level agent does not consider whether any of its teammates is performing any learning activity. A 1-level agent models its teammates as 0-level agents, and in general, a N-level agent models its teammates as (N-1)-level agents. This design principle is based on prior studies [25, 26, 27], which show that although higher-level agent models appear to be more powerful, the simplest 0-level agents can have better performance than 1-level and 2-level agents.

5. Agents should use a small state space to improve learning efficiency. All things being equal, an agent converges faster within a reduced state space. In addition, a small state space is also conducive to improving the scalability of a multi-agent system [3].

6. Agents should use a uniform state space so that knowledge sharing is possible across all agents. Knowledge sharing is also conducive to improving the learning efficiency, because the rate of updating the knowledge can be multiplied by the number of agents [18].

Following the above-mentioned design principles, we adopt a self-organizing neural model, namely *Temporal Difference - Fusion Architecture for Learning, COgnition, and Navigation (TD-FALCON)* [28, 24], for building the reinforcement learning (RL) agents in a multi-agent team. Based on Adaptive Resonance Theory (ART) [29, 30], FALCON is able to learn multi-channel mappings simultaneously across multi-modal input patterns, involving states, actions, and rewards, in an online and incremental manner [24]. In addition, unlike those gradient descent based backpropagation learning algorithms [14, 15, 16], the cognitive

nodes learned in FALCON is able to guarantee the stability of the learning performance. Integrating with the temporal difference (TD) algorithm, the TD-FALCON algorithm is able to function well in multi-agent environments with immediate as well as delayed rewards.

We begin our study by exploring a generic multi-agent environment which has no explicit requirement of collaboration and find that TD-FALCON agents can function and adapt well without knowing other agents' states and actions [31, 24]. For more complex multi-agent tasks, we develop various cooperative strategies to help TD-FALCON agents to adapt to the dynamic multi-agent environment. For each type of collaborative tasks, we use well-defined benchmark problems to compare the proposed cooperative strategies with the state-of-the-art approaches. Specifically, in the simplest multi-agent tasks that require explicit collaboration, we use a predator/prey navigation task as the benchmark to compare the proposed cooperative strategies with the non-cooperative counterparts [32, 31]. In more complex multi-agent environments with more input signals, we evaluate the proposed cooperative strategy using a herding game, showing that it outperforms the commonly-used joint action mechanism (JAM) [33, 34, 35]. In topology-based multi-agent systems (TMAS), we use a generic network routing problem to demonstrate that the proposed cooperative strategy can outperform classical techniques in the field of operation research [36].

1.4 Thesis Contributions

In this research work, we have successfully developed the TD-FALCON model for reinforcement learning and applied it in multi-agent cooperative learning environment. Additionally, several cooperative learning strategies are proposed so that the TD-FALCON agents can function and adapt efficiently in various multi-agent domains. The detailed contributions are described as follows.

1. We have successfully developed TD-FALCON and showed that it works well in a generic multi-agent environment with no explicit requirement of collaboration [8, 32].

Specifically, we conduct experiments based on a multi-agent minefield navigation task, wherein a number of autonomous vehicles (AVs) learn to navigate through obstacles to reach a stationary target (goal) within a specified number of steps. The experimental results on the navigation task 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 [31]. In comparison, the same level of performance is not attainable by traditional Q-learning agents [28, 31] based on multilayer feedforward neural networks trained with the backpropagation (BP) algorithm as the function approximator [37].

2. Subsequently TD-FALCON is applied in multi-agent tasks that require explicit collaboration. These include the predator/prey pursuit task, which calls for an explicit mechanism of cooperation for the agents to surround the prey from all directions [38]. For this pursuit task, we propose the CAT cooperative strategy, by using the distance from the center of agent team (CAT) to the prey to determine the team reward [32, 31]. Additionally, for pursuing high performance and efficiency of cooperative work, we use a compressed state representation to reduce the state space. Our experiments on the pursuit task show that the agents do not need to have in-depth knowledge of the other agents' underlying models to accomplish the task successfully [31]. Observing the environment and learning in an reactive manner enable the agents to predict the possible actions of other agents and potential outcomes of their own actions. In addition, our experiments on the pursuit game demonstrate that appropriate state representation and cooperative strategies play critical roles in producing superior performance while maintaining efficiency and scalability [32, 31].

3. We then deploy TD-FALCON in more diverse and complex multi-agent environments, where an agent may receive more signals from other agents. For example, in a herding game, the shepherds agents need to fulfil the task of encircling the sheep before forcing it to enter a corral within a grassland. In the herding game, we develop a neighboring-agent mechanism (NAM) strategy that only sensory inputs from the

neighboring agents are received and processed, further reducing the state space of an agent [39]. Our experiments in the herding game indicate that the NAM strategy can be implemented naturally and easily. Additionally, the experimental results also show that the NAM strategy has a higher level of efficiency than the CAT strategy as well as the traditional joint action mechanism [33, 34, 35], in terms of success rate, the number of steps to achieve the task, and the number of cognitive nodes.

4. We further extend the use of TD-FALCON to topology-based multi-agent systems (TMAS) domains, wherein each individual agent interacts only with its neighbors according to their spatial relationship. TMAS systems are well suited for problems with topological constraints, of which network routing (NR) is a classical example [36]. In the NR domain, each agent may have different state space, which can be rather large. To unify the state space across all agents in the NR domain, we have developed the TD-FALCON Binary Tree Formation (BTF) algorithm. Comparing with the TD-FALCON teams using the single agent strategy and n-ary ($n > 2$) tree formations, as well as a classical linear programming algorithm, TD-FALCON BTF team has produced superior performance in the experiments of a standard network routing problem [36]. In addition, the TD-FALCON BTF team demonstrates a high level of scalability under a varying level of network complexity and traffic volume.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives a literature study of the state-of-the-art reinforcement learning approaches as well as the multi-agent cooperative strategies. Chapter 3 presents the TD-FALCON architecture and algorithms, including the action selection policy and the value function estimation mechanism, as well as the experimental results and the complexity analysis. Chapter 4 describes the proposed methods and algorithms as well as the relevant experimental results on typical multi-agent environments, including the multi-agent minefield navigation task, the multi-predator/prey pursuit game and the herding game. Chapter 5 presents the TD-FALCON BTF strategy for the TMAS

domains and the experimental results of the strategy on the network routing task. Chapter 6 concludes the achievements and highlights the future work.

Chapter 2

Literature Study

In modern cognitive science, many have held the view that cognition is a process deeply rooted in the body's interaction with the world [40]. In other words, autonomous systems acquire intelligence through their interaction with the environment as a learning process [41].

As far as cooperative learning is concerned, there are two key approaches, namely *stochastic search* and *reinforcement learning (RL)*. Stochastic search operates by searching through the solution space and selecting from randomly generated candidate solutions. In the stochastic search literature, *Evolutionary Computation (EC)* is commonly adopted [3]. EC typically uses a fitness-oriented procedure to refine multiple agents' behaviors [3, 42]. Since EC is computationally intensive, it is not commonly used in real-time applications [5]. Our work, on the other hand, adopts the approach of reinforcement learning, a learning 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 [6, 7].

2.1 Reinforcement Learning

Reinforcement learning is based on a mathematical framework, namely Markov Decision Process (MDP). As described in the MDP paradigm [43], an autonomous agent typically op-

erates in a sense, act, and learn cycle (Figure 2.1). An autonomous system obtains sensory input from its environment representing the current state (**S**). Depending on the current state and its knowledge and goals, the system selects and performs the most appropriate action (**A**). Upon receiving the feedback in terms of rewards (**R**) from the environment, the system learns to adjust its behavior in the motivation of receiving positive rewards in the future.

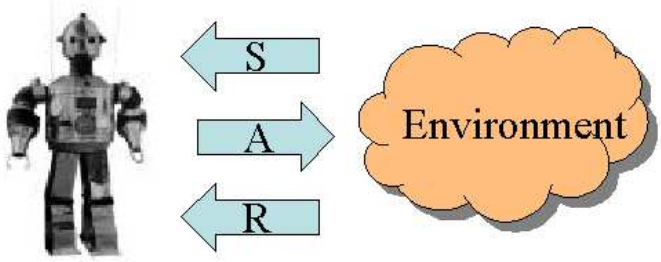


Figure 2.1: The embodied agent paradigm. An autonomous system receives sensory input (S) from its environment, executes an action (A), and receives feedback (R) from the environment.

In reinforcement learning, an agent aims to find a policy π, mapping states to actions, that maximizes some long-run measures of success. However, with a non-deterministic environment, taking the same action in a state in two different occasions may result in different states and/or different reinforcement values. In our work, to avoid the dilemma of the non-deterministic environments, we assume that the environment is deterministic, i.e., the probabilities of making state transitions or receiving specific reinforcement signals do not change over time [44].

2.1.1 Q-Learning

Classical approaches [45, 33] to the reinforcement learning problem generally 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, which

is a temporal difference method to estimate the accumulative future rewards (or costs) of performing an action in a given state. The Q-learning algorithm is elaborated as follows.

The problem model of Q-learning is composed of an agent, the set of states S and the set of actions A. After performing an action a ($a \in A$), the agent can move from one state to another state. Each state can give the agent a reward (a real number) or penalty (a negative reward). The goal of the agent is to maximize its total reward, by learning the optimal action for each state. Therefore, the Q-learning algorithm requires a value function to calculate the utility value of a state-action combination:

$$Q : S \times A \rightarrow \mathbb{R}. \tag{2.1}$$

Before learning is started, Q is initialized with a value chosen by the designer. Each time when the state is changed, the agent is given a reward, and a utility value is calculated for each combination of a state s ($s \in S$), and action a ($a \in A$). The core of the algorithm is a value iteration update, which assumes the old value and makes corrections repeatedly based on the new information. The value iteration update process can be denoted by the following equation:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha_t(s_t, a_t)}_{learning\ rate} \times [\overbrace{\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \underbrace{\max_a Q(s_{t+1}, a)}_{max\ future\ value}}^{expected\ discounted\ reward} - \overbrace{Q(s_t, a_t)}^{old\ value}], \tag{2.2}$$

where r_t is the reward given at time t, $\alpha_t(s_t, a_t)\,(0 < \alpha \leq 1)$ are the learning rates, which can be the same value for all state-action pairs, and $\gamma\,(0 \leq \gamma < 1)$ is the discount factor. Equation 2.2 can be transformed to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha_t(s_t, a_t)) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)]. \tag{2.3}$$

The drawback of the Q-learning algorithm is that agents must establish a table to store each tuple of state, action, and Q-values. Such requirement causes a scalability problem for large and/or continuous state and action spaces. Particularly, in a multi-agent system, an agent needs to keep track of its environment and the other agents. Thus the scalability problem is exacerbated.

2.1.2 Function Approximation Approaches

To tackle the scalability problem of Q-learning, many approaches based on function approximation have been proposed [12, 13]. Among those approaches, multi-layer perceptron (MLP) with the gradient descent based backpropagation (BP) learning algorithms has been widely used [14, 15, 16]. As shown in Figure 2.2, a MLP is a multi-layer feed-forward neural network, consisting of one layer (Layer 0, or the input layer) of input nodes, h ($h \geq 1$) layers (Layer 1 to h, or the hidden layers) of hidden nodes, and one layer (Layer $h+1$, or the output layer) of output nodes. Each pair of a node i in Layer m and a node j in Layer $m+1$ ($0 \leq m \leq h$) is connected and the connection is assigned with a weight marked as w_{ji}. Through the connection, the node i (the *predecessor*) is able to transfer its output o_i as an input to the node j (the *successor*).

A cycle of BP work flow is divided into two stages, namely *propagation* and *backpropagation*. During the propagation stage, the input nodes at first receive the sensory inputs from the environment and use them as their outputs. When a hidden or output node j in Layer m ($1 \leq m \leq h+1$) receives inputs from all the predecessors in Layer $m-1$, it computes the sum of the weighted inputs, generates the output from the sum by using a sigmoid function (a smoothed and differentiable threshold function), and transfers the output to all its successors in Layer $m+1$ if $1 \leq m < h+1$, or generates the system outputs if $m = h+1$. Assume that the node j has n predecessors in Layer $m-1$, using $\vec{w_j} = [w_{j1}, w_{j2}, \cdots, w_{jn}]$ as the weight vector and $\vec{x_j} = [x_{j1}, x_{j2}, \cdots, x_{jn}]$ as the input vector, the output o_j of the node j can be obtained by

$$o_j = \sigma(\vec{w_j} \cdot \vec{x_j}), \tag{2.4}$$

where σ is exactly the sigmoid function. A commonly used sigmoid function is

$$\sigma(y) = \frac{1}{1+e^{-y}}. \tag{2.5}$$

The nodes which use a sigmoid function to compute the output are named as sigmoid units. All hidden and output nodes in the BP network belong to sigmoid units.

After the system outputs are generated, the BP system enters the backpropagation stage.

At first, for each output node k, once its actual output value o_k and target output value t_k are known, the error term δ_k can be computed by

$$\delta_k = o_k(1 - o_k)(t_k - o_k). \tag{2.6}$$

Then the error terms of the output nodes are propagated backward (backpropagated) to update all weights. For a node i in Layer m $(0 \leq m \leq h)$, its error term δ_i can be computed by

$$\delta_i = o_i(1 - o_i) \sum_{s \in layer\, m+1} w_{si}\delta_s. \tag{2.7}$$

For a connection between the predecessor i and the successor j, if the error term δ_j is known, the weight w_{ji} can be updated by

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}, \tag{2.8}$$

and in terms of the stochastic gradient descent rule [46], the weight change term Δw_{ji} is computed by

$$\Delta w_{ji} = \eta \delta_j x_{ji}, \tag{2.9}$$

where η is the learning rate $(0 < \eta \leq 1)$ and x_{ji} is the input value from unit i to unit j.

The entire BP algorithm can be summarized in Table 2.1.

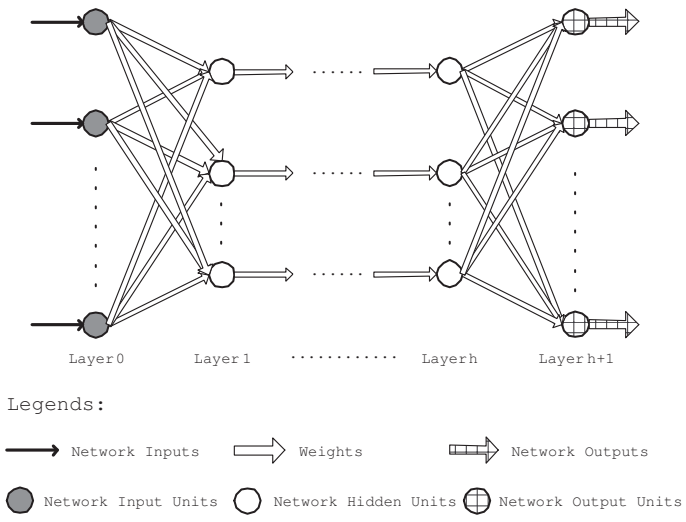


Figure 2.2: Artificial Neural Network.

1.	Repeat from Step 2 to Step 13 until the system converges.
	propagation stage
2.	The input layer receives the system input vector $\vec{x}$, and then outputs it to Layer 1.
3.	Repeat Step 4 to Step 5 for Layer $m = 1$ to $h + 1$.
4.	Repeat Step 5 for each node j in Layer m.
5.	Compute the output o_j of the node j, by using Equation 2.4.
6.	The network output layer generates the system output vector $\vec{o}$
	backpropagation stage
7.	Repeat Step 8 for each node k in the output layer.
8.	Calculate the error term δ_k for the node k, by using Equation 2.6.
9.	Repeat Step 10 to Step 13 for Layer $m = h$ to 0.
10.	Repeat Step 11 to Step 13 for each node i in Layer m.
11.	Calculate the error term δ_i for the node i, by using Equation 2.7.
12.	Repeat Step 13 for each node j in Layer $m + 1$.
13.	Update the weight w_{ji} by using Equation 2.8 and Equation 2.9.

Table 2.1: The Backpropagation Algorithm.

The BP algorithm avoids setting up a table to store each tuple of state, action, and Q-values and thus relieves the scalability issue. However, the BP algorithm is still facing three big issues. Firstly, the BP algorithm can not store all the previously learned patterns and thus the learning performance becomes very unstable. Secondly, to learn a pattern, a BP learner has to gone through many rounds of iterative learning, making it not suitable for a real-time learning environment. Finally, the BP neural network structure may need to be changed substantially (not limited to the input and output layers) to cope with different multi-agent problems.

On the basis of the backpropagation algorithm, Riedmiller and Braun [47, 48] propose the resilient propagation (RPROP) algorithm, which is also a type of backpropagation algorithms. The difference in the RPROP algorithm is that it updates the weights by using the signs of two successive derivatives [49] in place of the magnitudes of the derivatives as the BP algorithm uses.

In detail, RPROP uses the following equation to replace Equation 2.9 in Step 13 of Table 2.1 to compute the weight change term (at round t of a RPROP work flow):

$$\Delta w_{ji}^{(t)} = \begin{cases} -\Delta_{ji}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} > 0 \\ \Delta_{ji}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} < 0 \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

where E is the standard error of the system outputs and $\Delta_{ji}^{(t)}$ is the weight-specific update value at round t. Equation 2.10 indicates that the *direction* of the weight update is purely determined by the *sign* of the partial derivative of the weight w_{ji} at round t.

The *size* of the weight update is determined by the update value at round t:

$$\Delta_{ji}^{(t)} = \begin{cases} \eta^{+} * \Delta_{ji}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} * \frac{\partial E}{\partial w_{ji}}^{(t-1)} > 0 \\ \eta^{-} * \Delta_{ji}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} * \frac{\partial E}{\partial w_{ji}}^{(t-1)} < 0 \\ \Delta_{ji}^{(t)} & \text{otherwise,} \end{cases} \tag{2.11}$$

where $0 < \eta^{-} < 1 < \eta^{+}$ and $\Delta_{ji}^{(t-1)}$ is the update value of w_{ji} at round $t-1$. Equation 2.11 conveys the following ideas of the RPROP algorithm: (1) when the sign of the partial

derivative of the weight w_{ji} is changed, meaning that the weight update at the last round is too great and thus the local minimum has been overstepped, the weight update value Δ_{ji} should be decreased by multiplying the learning rate η^-, in order for the system to converge to the local minimum; (2) when the sign of the partial derivative of the weight w_{ji} is remained, meaning that the weight update is still on the same direction and the local minimum has not been reached yet, the weight update value Δ_{ji} should be increased by multiplying the learning rate η^+, in order to accelerate the convergence of the RPROP system.

In comparison with the standard error backpropagation algorithm described in Table 2.1, RPROP offers faster learning, lower computation cost, higher robustness in parameter choice and faster convergence rate. However, RPROP has not overcome the inherent limitations of the gradient descent methods, including the instability in learning performance and iterative learning. As a result, the improvement of the RPROP algorithm is generally limited.

There are also other developments in the literature of function approximation Q-learning. Tesauro [50] proposes the Hyper-Q learning, in which the values of mixed strategies instead of the base actions are learned and other agents' strategies are estimated from the observed actions by using Bayesian inference. However, with function approximation, the convergence of Q-learning becomes substantially more difficult. Kose *et al.* [12] use Cerebellar Model Articulation Controller (CMAC) for function approximation and state generalization in robot soccer games, because of its efficiency in learning and operation. They use a market-driven auction mechanism to select the most appropriate agent to undertake a subtask by comparing the cost of each agent. However, the market-driven approach has the disadvantage that the system may converge to a suboptimal solution. Kononen [13] proposes a gradient-based method, which is a combination of value function approximation and direct policy gradients with the Value And Policy Search (VAPS) framework for multi-agent reinforcement learning. By including terms corresponding to action choices and rewards of both agents into the history, Kononen extends the VAPS framework to a two-agent case. However, the VAPS structure has the problem that even if there are only

small variations in the other agent's value function, an agent may have large changes in its strategies and the future values of the states. As a result, the two agents in the VAPS framework can hardly converge.

Although the above-mentioned function approximation approaches can relieve the scalability issue in some degree, they generally have the problems of the unstable learning performance and the use of iterative learning. Since there is no way to store all the patterns using a function approximation method, learning new patterns may erase the previously learned knowledge. Therefore, as Yang and Gu [33] point out, Q-learning with function approximation can be very unstable. Moreover, most function approximation methods depend on iterative learning [51], and thus the resultant multi-agent reinforcement learning systems may not work well in real-time environments.

2.1.3 FALCON and TD-FALCON

Tan [52] proposes the Fusion Architecture for Learning, COgnition, and Navigation (FALCON), based on multi-channel Adaptive Resonance Associative Map (multi-channel ARAM) [53]. FALCON is a 3-channel ARAM, which consists of a sensory field representing the current state, an action field representing the available actions, and a reward field representing the values of the feedback received. Inherited from the ARAM structure, FALCON uses a category field to store the cognitive nodes developed during the learning process, avoiding the unstable and iterative learning occurring in traditional function approximation approaches [12, 13, 14, 15, 16].

The FALCON system employs fuzzy ART operations [54] to emulate the activities of sense, act, and learn, operating in either the predicting or the learning mode. In the prediction mode, by receiving input values in one or more fields, FALCON predicts the values for the remaining fields. Under the learning mode, FALCON selects a winner cognitive node to complete the learning process using template matching, template learning and node creation [32, 31].

Basically there are two types of FALCON models, namely the reactive FALCON model

(R-FALCON) and Temporal Difference (TD) FALCON. R-FALCON learns a function mapping from the states to the actions directly, based on the reinforcement signal received after taking each action [24]. However, such a reinforcement signal may not be immediately available in a practical environment.

In contrast to R-FALCON, TD-FALCON [8, 24, 31] incorporates the temporal difference (TD) methods to estimate and learn value functions, which show the goodness of taking an action in a given state. The general sense-act-learn TD-FALCON algorithm has three stages, namely *value prediction*, *action selection* and *value function estimation* [32, 31, 24]. In the value prediction stage, the FALCON network operates in prediction mode and predicts the value of performing each available action. During the action selection process, an agent picks an action to take in a given state from the set of the available actions. An effective action selection policy should have a balance between *exploitation*, i.e., sticking to the best actions believed, and *exploration*, i.e., trying out other inferior and less familiar actions, so as to maximize the reward over time. The key component of the TD-FALCON is the iterative estimation of value function, wherein a temporal difference equation is used to implement the multiple-step prediction mechanism.

2.2 Multi-Agent Cooperative Strategies

In multi-agent cooperative learning environments, multiple agents cooperate to solve a given task jointly and/or to maximize certain utility function through their interactions [3].

Existing multi-agent cooperative strategies can be broadly classified into *team learning* and *concurrent learning* [3]. Team learning is a centralized learning algorithm, which involves a single learner to carry out the learning behaviors of all the agents. A team learning algorithm requires all resources and information to be collected by a single learner, leading to the increase of the communication load in an inherently distributed system [55]. A commonly used team learning strategy in traditional work of multi-agent cooperative learning is the joint action mechanism (JAM) [33, 34, 35], wherein the sensory inputs and

actions from all agents are combined into a single team learner, and a unified reward is given to the whole team after a combined action is performed. The mathematical model of the JAM is presented below.

Assume that there are n agents, namely $AGT_1, AGT_2, \cdots, AGT_n$, in a multi-agent system. Each agent $AGT_i\,(1 \leq i \leq n)$ has p_i states and q_i actions. The state space of AGT_i can be denoted by $S^i = \{s^{i,1}, s^{i,2}, \cdots, s^{i,p_i}\}$, and the action space of AGT_i can be denoted by $A^i = \{a^{i,1}, a^{i,2}, \cdots, a^{i,q_i}\}$. According to the joint action mechanism, we assign a single team learner L, which simply combines states and actions across all n agents. L has the state space $S = S^1 \times S^2 \times \cdots \times S^n$ and the action space $A = A^1 \times A^2 \times \cdots \times A^n$. As a result, the size of the state space of L is

$$|S| = p_1 p_2 \cdots p_n = \prod_{i=1}^{n} p_i, \tag{2.12}$$

and the size of the action space of L is

$$|A| = q_1 q_2 \cdots q_n = \prod_{i=1}^{n} q_i. \tag{2.13}$$

The joint action mechanism is able to provide a uniform formation of state and action spaces, and spare the development of cooperative strategies among agents by issuing unified commands from the team learner L. Nevertheless, from Equation 2.12 and 2.13, the state and action spaces of a JAM team will grow exponentially with respect to the number of involved agents, causing the combinatorial explosion [56].

Using concurrent learning, each agent is equipped with a learner to do the learning simultaneously [57]. Concurrent learning generally involves three main considerations, namely *reward assignment* [58], *dynamics of learning* [59] and *modeling of other agents* [35].

In terms of reward assignment, typically there are methods that use *global rewards* [60], *local rewards* [61] and/or *observational reinforcement rewards* [62]. 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 reward by observing and imitating other agents' behaviors.

In terms of the dynamics of learning, there are mainly two scenarios, namely *fully-cooperative* [63] and *general-sum game* [64]. In a fully-cooperative scenario, the rewards received by agents are correlated, and so increasing one agent's reward implies increasing other agents' rewards. Contrarily, in the case of the general-sum game, increasing one agent's reward does not necessarily result in increasing the reward of the whole team. The general-sum game may lead to highly non-cooperative situations.

In terms of modeling of other agents, there are *0-level* modeling agents, *1-level* modeling agents, *2-level* modeling agents and so on [65]. Generally, a 0-level agent does not consider whether any of its teammates is performing any learning activity, but just regards them as part of a dynamic environment. A 1-level agent models its teammates as 0-level agents, and in general, a N-level agent models its teammates as (N-1)-level agents. M. Mundhe and S. Sen [25], S. Sen and M. Sekaran [26] as well as S. Sen and M. Sekaran and J. Hale [27] have found that 0-level agents have better performance than 1-level and 2-level agents.

Existing works on multi-agent systems have explored the various combinations of reward assignment, dynamics of learning, and/or agent modeling. For example, Littman [17] proposes a minimax Q-learning algorithm to update the V values in two-player zero-sum stochastic games. Tan [18] shows that 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. To extend to general-sum stochastic games, Hu and Wellman [19] propose the Nash-Q algorithm to update the V values based on some Nash equilibrium. Although the Nash-Q algorithm can work in general-sum environments, the solution is restricted to only two players and to the condition that the other agent learns the same Nash equilibrium.

Agogino and Tumer [66] propose the QUICR-learning to divide a global reward into many agent-specific rewards in a large-scale multi-agent system. The proposal has two properties: 1) agents increasing their agent specific rewards tend to increase the global reward, 2) an agent's action has a great influence on its agent-specific reward. The solution however has the disadvantage that the QUICR system is based on $TD(0)$, which can not work without immediate reward. By exploiting the relationship to standard Q-learning, Lauer

and Riedmiller [67] present the full-lists approach to represent the expected costs-to-go for all agents in list-based data structures in place of tables to reduce communications between agents. However, the full-lists approach has the problem of combinatorial explosion. Due to this issue, they adopt the reduced-lists approach instead, wherein a threshold is set to reduce the number of entries in each list, but at the expense of missing the possibly optimal solutions.

In general, although there is a great development in cooperative strategies to enable multiple agents to converge to some optimal solutions, those approaches are still restricted to a limited number of players and/or certain conditions. For example, the minimax Q-learning [17] is limited to two agents with diametrically opposed goals, the Nash-Q algorithm [19] is restricted to only two players and to the condition that the other agent learns the same Nash equilibrium, and the QUICR system [66] can only work under immediate reward. The reason is that the approaches can not cope with the increasing complexity of multi-agent environments effectively, if more agents are involved.

Chapter 3

Reinforcement Learning with Self-Organizing Neural Architectures

To overcome the limitations of the traditional function approximation approaches in Q-learning, this thesis work proposes the Fusion Architecture for Learning, COgnition, and Navigation (FALCON) [52] as the model in multi-agent reinforcement learning. Based on Adaptive Resonance Theory (ART) [29, 30], FALCON is able to perform online and incremental learning in a real-time multi-agent environment. Incorporating with the temporal difference (TD) algorithm, FALCON is extended to TD-FALCON, which can function and adapt well in a multi-agent environment with immediate and/or delayed reward.

3.1 Background and Motivations

Among the self-organizing neural networks, Self-Organizing Map (SOM) is has been typically used for reinforcement learning. Using a localized representation, SOM has the advantage of more stable learning, compared with backpropagation neural networks based on distributed representation. However, SOM remains an iterative learning system, which requires many rounds to learn the clusters of state and action patterns. In addition, most applications of SOM are limited to low dimensional state and action spaces [68, 69], because

the performance of SOM noticeably deteriorates due to over generalization with the increase of dimensionality.

As another class of self-organizing neural networks, Adaptive Resonance Theory (ART) [29] has very distinct characteristics compared with SOM. The most remarkable advantage of ART is that through an unique code stabilizing and dynamic network expansion mechanism, ART models are capable of learning recognition categories of multi-dimensional mappings of input patterns in an online and incremental manner. They are thus suitable mechanisms for autonomous systems to learn value functions. However, whereas various models of ART and their predictive (supervised learning) versions [70, 71] have been widely applied to pattern analysis and recognition tasks, there have been very few attempts to use ART-based networks for reinforcement learning and building autonomous systems. Ninomiya [72] proposes a hybrid architecture that couples a supervised ART system known as ARTMAP-IC with a temporal difference reinforcement learning method. However, the states and actions in the reinforcement module are exported from ARTMAP-IC and the two learning systems function independently. The redundancy leads to unnecessarily long processing time in selecting actions as well as learning value functions.

By inheriting the fast and stable learning characteristics from ART models, The TD-FALCON system proposed in this thesis is able to learn the Q-value function in a much faster pace, with the stability of learning retained.

3.2 FALCON Architecture

The proposed FALCON architecture is based on multi-channel Adaptive Resonance Associative Map (multi-channel ARAM) [53] or fusion ART [28], which is an extension of predictive Adaptive Resonance Theory (ART) networks. Whereas predictive neural network models, such as ARTMAP network [70] and ARAM [71], learn multi-dimensional mappings between the input and output patterns, multi-channel ARAM formulates cognitive codes associating multi-modal patterns across multiple input channels.

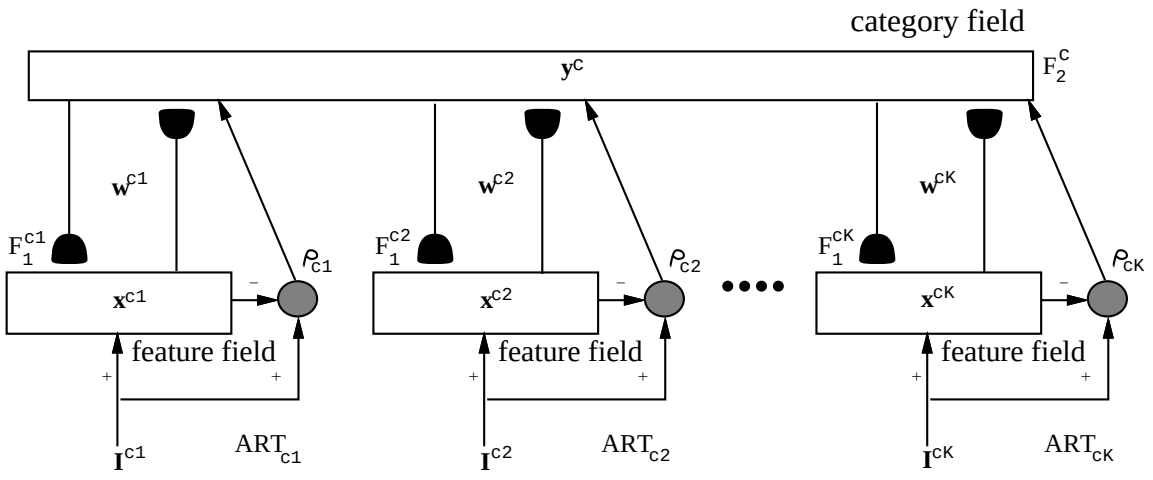


Figure 3.1: The architecture of multi-channel ARAM.

A multi-channel ARAM model consists of a category field F_2^c and a number of input representation fields $F_1^{c1}, F_1^{c2}, \ldots, F_1^{cK}$ (Figure 3.1). 2-channel ARAM is equivalent to ARTMAP. If only one channel is present, multi-channel ARAM reduces to ART. For reinforcement learning, FALCON makes use of a 3-channel architecture, consisting of a sensory field for representing the current state, an action field for representing the available actions, and a reward field for representing the values of the feedback received from the environment.

3.2.1 FALCON Structure

The generic network dynamics of FALCON based on 3-channel ARAM, which employs fuzzy ART operations [73], is described below.

Input vectors: Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where $s_i \in [0, 1]$ indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where $a_i \in [0, 1]$ indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ and $\bar{r} = 1 - r$. 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 [74].

Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector. 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 1's. When an *uncommitted* node is selected to learn an association, it becomes *committed*.

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

To emulate the activities of sense, act, and learn, FALCON network operates in one of the two modes, namely predicting and learning. The detailed algorithm is described in Section 3.2.2 and Section 3.2.3.

3.2.2 Predicting

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, 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:

$$T_j^c = \sum_{k=1}^{3} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}, \tag{3.1}$$

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

$$T_J^c = \max\{T_j^c : \text{for all } F_2^c \text{ node } j\}. \tag{3.2}$$

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

$$\mathbf{x}^{ck}(\text{new}) = \mathbf{x}^{ck}(\text{old}) \wedge \mathbf{w}_J^{ck}. \tag{3.3}$$

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

3.2.3 Learning

For learning value functions, all the state, action, and reward vectors are presented to FALCON. Therefore, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R}$. Under the learning mode, 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:

$$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}. \tag{3.4}$$

When a resonance occurs, the template learning ensues, as defined below. 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 0 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:

$$\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})}). \tag{3.5}$$

For an uncommitted node J, the learning rate β^{ck} is typically set to 1. For committed nodes, β^{ck} can remain as 1 for fast learning or below 1 for slow learning in a noisy environment.

Watch the formatting of F_2^c.

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.

3.2.4 Reactive FALCON

The reactive FALCON model (R-FALCON) acquires an action policy directly by learning the mapping from the current states to the desirable actions. For completeness, we provide a summary of the R-FALCON dynamics based on the generic FALCON algorithm below. Please refer to [52] for a detailed description.

3.2.4.1 From Sensory to Action

Upon an input presentation, the activity vectors are initialized as $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{N}$, and $\mathbf{x}^{c3} = (1, 0)$, where $N_i = 1$ for all i. Setting the reward vector to $(1, 0)$ enables the selection of a cognitive node with the maximum reward value for a given state.

With the activity vector values (specifically the state vector $\mathbf{S}$ and the reward vector), the system performs code competition and selects an action based on the output activities of action vector $\mathbf{A}$ as follows.

Action selection: Upon selecting a winning F_2^c node J, the chosen node J performs a readout of its weight vector to the action field F_1^{c2} such that

$$\mathbf{x}^{c2} = \mathbf{x}^{c2(\text{old})} \wedge \mathbf{w}_J^{c2}. \tag{3.6}$$

An action a_I is then chosen such that

$$x_I^{c2} = \max\{x_i^{c2} : \text{for all } F_1^{c2} \text{ node } i\}. \tag{3.7}$$

3.2.4.2 From Feedback to Learning

Upon receiving a feedback from its environment after performing the action a_I, the system adjusts its internal representation based on the following principles. Given a reward (positive feedback), the agent learns that the chosen action executed in a given state will result in a favorable outcome. Therefore, the system learns to associate the state vector $\mathbf{S}$, the action vector $\mathbf{A}$, and the reward vector $\mathbf{R}$. During input presentation, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R}$.

Conversely, if a penalty is received, there is a reset of action and the agent learns the mapping among the state vector $\mathbf{S}$, the complement of action vector $\bar{\mathbf{A}}$ where $\bar{a}_i = 1 - a_i$ for all i, and the complement of reward vector $\bar{\mathbf{R}}$ where $\bar{r}_i = 1 - r_i$ for all i. During input presentation, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \bar{\mathbf{A}}$, and $\mathbf{x}^{c3} = \bar{\mathbf{R}}$.

R-FALCON then proceeds to learn the association among the activity vectors of the three input fields using the learning algorithm as described in Section 3.2.3.

3.3 TD-FALCON

Note that R-FALCON learns based on the feedback obtained after performing each action. In a practical environment, an autonomous agent may not immediately receive feedback upon an action. Typically, it may take a long sequence of actions before a reward or a penalty is finally given. This is known as a temporal credit assignment problem [75, 15] in which we need to estimate the credit/value of an action based on what it will lead to eventually.

In contrast to R-FALCON that learns a function mapping states to actions directly, TD-FALCON incorporates Temporal Difference (TD) methods to estimate and learn value functions, specifically, functions of action-state pairs $Q(s, a)$ that indicates the goodness for

a learning system to take a certain action a in a given state s. Such value functions guide the action selection mechanism, or *the policy*, to achieve a balance between exploration and exploitation, so as to maximize the reward over time. A key advantage of using TD methods is that they can be used for multiple-step prediction problems, in which the merit of an action can only be known after several steps into the future.

3.3.1 TD-FALCON Algorithm

The general sense-act-learn algorithm of TD-FALCON is summarized in Table 3.1. 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.

3.3.2 Value Prediction

Given the current state s, the FALCON network is used to predict the value of performing each available action a in the action set $\mathcal{A}$ based on the corresponding state vector $\mathbf{S}$ and action vector $\mathbf{A}$. Upon input presentation, the FALCON activity vectors are initialized as $\mathbf{x}^{c1} = \mathbf{S} = (s_1, s_2, \ldots, s_n)$ where $s_i \in [0, 1]$ indicates the value of sensory input i, $\mathbf{x}^{c2} = \mathbf{A} = (a_1, a_2, \ldots, a_n)$ where $a_I = 1$ if a_I corresponds to the action a and $a_i = 0$ for $i \neq I$, and $\mathbf{x}^{c3} = (1, 1)$.

With the activity vector values, the system performs code activation and code competition as described in Section 3.2.2. Upon selecting a winning F_2^c node J, the chosen node J performs a readout of its weight vector to the reward field F_1^{c3} such that

$$\mathbf{x}^{c3} = \mathbf{x}^{c3}(\text{old}) \wedge \mathbf{w}_J^{c3}. \tag{3.8}$$

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 **S** and **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 (**S**, **A**, and **R**) to FALCON for learning.
7. Update the current state by s=s'.
8. Repeat from Step 2 until s is a terminal state.

Table 3.1: Generic dynamics of TD−FALCON.

The Q-value of performing the action a in the state s is then given by

$$Q(s, a) = \frac{x_1^{c3}}{\sum_i x_i^{c3}}. \tag{3.9}$$

If node J is uncommitted, $\mathbf{x}^{c3} = (1, 1)$ and thus the predicted Q-value is 0.5.

3.3.3 Action Selection Policy

Action selection policy refers to the strategy used to pick an action from the set of the available actions for an agent to take in a given state. It is the policy referred to in step 3 of the TD-FALCON algorithm described in Table 3.1. The simplest action selection policy is to pick the action with the highest value predicted by the TD-FALCON network. However, a key requirement of autonomous agents is to explore the environment. If an agent keeps selecting the optimal action that it believes, it will not be able to explore and discover better

alternative actions. There is thus a fundamental tradeoff between *exploitation*, i.e., sticking to the best actions believed, and *exploration*, i.e., trying out other seemingly inferior and less familiar actions. Two policies, designed to achieve a balance between exploration and exploitation, are presented below.

3.3.3.1 ϵ-greedy Policy

The ϵ-greedy policy selects the action with the highest value with a probability of $1-\epsilon$, where ϵ is a constant between 0 and 1, and takes a random action, with probability ϵ [76]. In other words, the policy will pick the action with the highest value with a total probability of $1-\epsilon+\frac{\epsilon}{|A(s)|}$ and any other action with a probability of $\frac{\epsilon}{|A(s)|}$, where $A(s)$ denotes the set of the available actions in a state s and $|A(s)|$ denotes the number of the available actions.

With a fixed ϵ value, the agent will always explore the environment with a fixed level of randomness. In practice, it may be beneficial to have a higher ϵ value to encourage the exploration of paths in the initial stage and a lower ϵ value to optimize the performance by exploiting familiar paths in the later stage. A decayed ϵ-greedy policy is thus proposed to gradually reduce the value of ϵ over time. The decayed rate is typically inversely proportional to the complexity of the environment as a more complex environment with a larger input and action space will take a longer time to explore.

3.3.3.2 Softmax Policy

Under the softmax policy, the probability $p(a_i)$ of choosing an action a_i is given by the following expression:

$$p\left(a_i\right)=\frac{e^{Q_t(a_i)/\tau}}{\sum_{a_j=1}^{n} e^{Q_t(a_j)/\tau}} \tag{3.10}$$

where τ is a positive parameter called the temperature. At a high temperature, all actions are equally likely to be taken, whereas at a low temperature, the probability of taking a specific action is more dependent on the value estimate of the action.

3.3.4 Value Function Estimation

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

$$\Delta Q(s,a) = \alpha TD_{err} \tag{3.11}$$

where $\alpha \in [0,1]$ is the learning parameter and TD_{err} is a function of the current Q-value predicted by TD-FALCON and the Q-value newly computed by the TD formula. Two distinct Q-value updating rules, namely Q-learning and SARSA, are described below.

3.3.4.1 Q-learning

Using the Q-learning rule, the temporal error term is computed by

$$TD_{err} = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{3.12}$$

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'. It is important to note that the estimation of $\max_{a'} Q(s',a')$ is read out from the TD-FALCON network by examining all possible actions for the next state and not by another reinforcement learning system. The update rule is applied to all states that the agent traverses. With value iteration, the value function $Q(s,a)$ is expected to converge to $r + \gamma \max_{a'} Q(s',a')$ over time.

In general, there is no restriction to the value of reward r and thus the value function $Q(s,a)$. However, in TD-FALCON and many other pattern associators, it is commonly assumed that all input values are bounded between 0 and 1. A simple solution to this problem is to apply a threshold function to the Q-values such that

$$Q(s,a) = \begin{cases} 1 & \text{if } Q(s,a) > 1 \\ 0 & \text{if } Q(s,a) < 0 \\ Q(s,a) & \text{otherwise} \end{cases} \tag{3.13}$$

The threshold function, though simple, provides a sufficiently good solution if the reward value r itself is bounded within a range, say between 0 and 1.

Instead of using the threshold function, Q-values can be normalized by incorporating appropriate scaling terms into the Q-learning updating equation directly. The Bounded Q-Learning rule is given by

$$\Delta Q(s,a) = \alpha TD_{err}\left(1 - Q\left(s,a\right)\right). \tag{3.14}$$

By introducing the scaling term $1 - Q\left(s,a\right)$, the adjustment of Q-values will be self-scaled so that they will not be increased beyond 1. The learning rule thus provides a smooth normalization of the Q-values. If the reward value r is constrained between 0 and 1, we can guarantee that the Q-values will be bounded between 0 and 1.

Lemma: Bounded Q-learning Rule Given that $0 \leq r \leq 1$, $0 \leq \alpha \leq \frac{1}{2}$, $\gamma \leq 1$, and initially $0 \leq Q(s,a) \leq 1$, the Bounded Q-learning rule

$$\Delta Q(s,a) = \alpha TD_{err}(1 - Q(s,a)) \tag{3.15}$$

where

$$TD_{err} = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{3.16}$$

ensures that the Q-values will be bounded between 0 and 1, i.e., $0 \leq Q(s,a) \leq 1$, and that when learning ceases, the Q-values will converge to either $r + \gamma \max_{a'} Q(s',a')$ if $r + \gamma \max_{a'} Q(s',a') < 1$, or 1 otherwise.

Proof: The proof of the lemma consists of three parts as follows.

Part I: To prove that $Q(s,a) \leq 1$, we show the new Q-values computed by the updating rule will not be greater than 1.

$$\begin{aligned} Q^{(\text{new})}(s,a) &= Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)(1 - Q(s,a)) \\ &\leq Q(s,a) + 0.5(1 + 1 \times 1 - 0)(1 - Q(s,a)) \\ &\leq Q(s,a) + 1 - Q(s,a) \\ &\leq 1 \end{aligned}$$

Part II: To prove that $Q(s,a) \geq 0$, we show the new Q-values computed by the updating

rule will not be smaller than 0.

$$\begin{aligned} Q^{(\text{new})}(s,a) &= Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)(1 - Q(s,a)) \\ &\geq Q(s,a) + \alpha\left(0 + 0 \times 0 - Q(s,a)\right)(1 - Q(s,a)) \\ &\geq Q(s,a) + \alpha\left(-Q(s,a)\right)(1 - Q(s,a)) \\ &\geq \alpha Q^2(s,a) + (1-\alpha)Q(s,a) \\ &\geq 0 \end{aligned}$$

Part III: When learning ceases, we have $\Delta Q(s,a) = 0$. This implies that either

$$r + \gamma \text{max}_{a'} Q(s',a') - Q(s,a) = 0 \tag{3.17}$$

or

$$1 - Q(s,a) = 0. \tag{3.18}$$

Therefore, we have either $Q(s,a) = r + \gamma \text{max}_{a'} Q(s',a')$ or $Q(s,a) = 1$.

[end of proof].

As Q-values are estimates of the discounted sums of future rewards in a given state, our requirement for Q-values to be bounded within the range of 0 and 1 imposes certain restriction on the types of problems TD-FALCON can handle directly. In cases where the discounted sums of future rewards fall significantly outside $[0, 1]$, TD-FALCON may lack the sensitivity to learn the Q-values accurately.

3.3.4.2 SARSA

Using the SARSA updating rule, the temporal error term is computed by

$$TD_{err} = r + \gamma Q(s',a') - Q(s,a) \tag{3.19}$$

where r is the immediate reward signal, $\gamma \in [0, 1]$ is the discount parameter, and $Q(s', a')$ denotes the estimated value of the next state s'. Unlike Q-learning, SARSA does not have

a separate estimation policy. Whereas Q-learning estimates future reward as a function of the discounted maximum possible reward of taking an action a' from the next state s', SARSA simply estimates the future reward using its behavior policy with a discounted factor given by $\gamma Q(s, a)$. Consequently, SARSA is said to be an on-policy as it estimates value functions based on the actions that it has taken. With value iteration, the value function $Q(s, a)$ is expected to converge to $r + \gamma Q(s', a')$.

As the value range of TD_{err} for SARSA is the same as that for Q-Learning, the normalization techniques derived for Q-learning (Equation 3.14) are applicable to SARSA. Following the Bounded Q-learning rule, the Bounded SARSA learning rule is given by

$$\Delta Q(s, a) = \alpha \left(r + \gamma Q\left(s', a'\right) - Q\left(s, a\right)\right)\left(1 - Q\left(s, a\right)\right). \tag{3.20}$$

3.4 Experimental Results

To test the performance of the TD-FALCON algorithm in a real-time environment, we arrange a series of experiments based on a minefield simulation task. These experiments are divided into three groups: (1) performance comparison among TD-FALCON teams using various Q-value updating rules, including R-FALCON, Q-FALCON (FALCON with Threshold Q-learning), BQ-FALCON (FALCON with Bounded Q-learning), S-FALCON (FALCON with Threshold SARSA), and BS-FALCON (FALCON with Bounded SARSA), in an environment with immediate evaluative feedback; (2) performance comparison among these TD-FALCON teams in an environment with delayed evaluative feedback; (3) make comparison with the traditional gradient descent based backpropagation (BP) Q learner, as described in Table 2.1.

3.4.1 The Minefield Simulation Task

The minefield simulation task used in the experiments is similar to the underwater navigation and mine avoidance domain developed by Naval Research Lab (NRL) [77]. The objective of an agent is to navigate through a minefield to a randomly selected target

position in a specified time frame without hitting a mine.

For experimentation, we develop a software simulator for the minefield simulation task. The simulator allows a user to specify the size of the minefield as well as the number of mines in the field. Our experiments are based on a 16 by 16 minefield containing 10 mines. In each trial, an autonomous vehicle (AV) starts at a randomly chosen position in the field, and repeats the cycles of sense, act, and learn. A trial ends when the agent reaches the target (success), hits a mine (failure), or exceeds 30 sense-act-learn cycles (out of time). The target and the mines remain stationary during the trial.

Minefield navigation with mine avoidance is a non-trivial task. As the configuration of the minefield is generated randomly and changes over trials, the system needs to learn strategies that can be carried over across experiments. In addition, the system has a rather coarse sensory capability with a 180 degree forward view based on five sonar sensors. For each direction i, the sonar signal is measured by

$$s_i = \frac{1}{d_i}, \tag{3.21}$$

where d_i is the distance to an obstacle (that can be a mine or the boundary of the minefield) in the i direction. Other input attributes of the sensory (state) vector include the bearing of the target from the current position. In each step, the system can choose one of the five possible actions, namely move left, move diagonally left, move straight ahead, move diagonally right, and move right.

3.4.2 Learning with Immediate Reinforcement

We first consider the problem of learning the minefield simulation task with immediate evaluative feedback. The reward scheme is described as follows: At the end of a trial, a reward of 1 is given when the AV reaches the target. A reward of 0 is given when the AV hits a mine. At each step of the trial, an immediate reward is estimated by computing a utility function

$$utility = \frac{1}{1 + r_d}, \tag{3.22}$$

where r_d is the remaining distance between the current position of the AV and the target position. Even when the AV runs out of time, the reward can still be computed using the utility function based on the remaining distance to the target.

We experiment with R-FALCON that learns the state-action policy directly and four types of TD-FALCON models, namely Q-FALCON and BQ-FALCON based on Threshold Q-learning and Bounded Q-learning respectively, as well as S-FALCON and BS-FALCON based on Threshold SARSA and Bounded SARSA respectively. Each FALCON system consists of 18 nodes in the sensory fields (representing 5x2 complement-coded sonar signals and eight target bearing values), five nodes in the action field, and two nodes in the reward field (representing the complement-coded function value).

All FALCON systems use a standard set of parameter values as shown in Table 3.2. The choice parameters are used in the choice function (Equation 3.1) in selecting cognitive nodes. Using a larger choice value generally improves the predictive performance of the system but increases the number of cognitive nodes created. The learning rate parameters β^{ck} for $k = 1, 2, 3$ are set to 1.0 for fast learning. Decreasing the learning rates slows down the learning process, but may produce a smaller set of better quality cognitive nodes and thus lead to slightly better predictive performance. The contribution parameters γ^{c1} and γ^{c2} are set to 0.5 as TD-FALCON selects a cognitive node based on the input activities in the state and action fields. The baseline vigilance parameters $\bar{\rho}^{c1}$ and $\bar{\rho}^{c2}$ are set to 0.2 for a marginal level of match criterion on the state and action spaces so as to encourage generalization. The vigilance of the reward field $\bar{\rho}^{c3}$ is fixed at 0.5 for a stricter match criterion. Increasing the vigilance values generally increases the predictive performance with the cost of creating more cognitive nodes. For the temporal difference learning rules, the learning rate α is fixed at 0.5 to allow a modest pace of learning while retaining stability. The discount factor γ is set to 0.1 to favour the direct reward signals available. The initial Q-value, used when TD-FALCON selects an uncommitted node during prediction, is set to 0.5, corresponding to a weight vector of (1,1). For action selection policy, the decayed ϵ-greedy policy is used with ϵ initialized to 0.5 and decayed at a rate of 0.0005 per trial, until ϵ drops to 0.005. This implies that the system will have a low chance to explore new

FALCON Parameters	
Choice parameters $(\alpha^{c1}, \alpha^{c2}, \alpha^{c3})$	$0.1, 0.1, 0.1$
Learning rates $(\beta^{c1}, \beta^{c2}, \beta^{c3})$	$1.0, 1.0, 1.0$
Contribution parameters $(\gamma^{c1}, \gamma^{c2}, \gamma^{c3})$	$0.5, 0.5, 0.0$
Vigilance parameters $(\rho^{c1}, \rho^{c2}, \rho^{c3})$	$0.2, 0.2, 0.5$
Temporal Difference Learning Parameters	
TD learning rate α	0.5
Discount factor γ	0.1
Initial Q-value	0.5
ϵ-greedy Action Policy Parameters	
Initial ϵ value	0.5
ϵ decay rate	0.0005

Table 3.2: TD-FALCON parameters for learning with immediate rewards.

moves after around 1000 trials.

Figure 3.2 summarizes the performance of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON in terms of success rates averaged at 200-trial intervals over 3000 trials across ten sets of experiments. We can see that the success rates of all systems increase steadily right from the beginning. Among all, R-FALCON is the fastest, achieving 90 percent at 600 trials. Nevertheless, beyond 1000 trials, all TD-FALCON systems can achieve over 90 percent success rates. In the long run, R-FALCON and all four TD-FALCON systems achieve roughly the same level of performance.

To evaluate in quantitative terms how well a system traverses from a starting position to the target, we define a measure called *normalized step* given by

$$step_n = \frac{step}{s_d}, \tag{3.23}$$

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 1 means

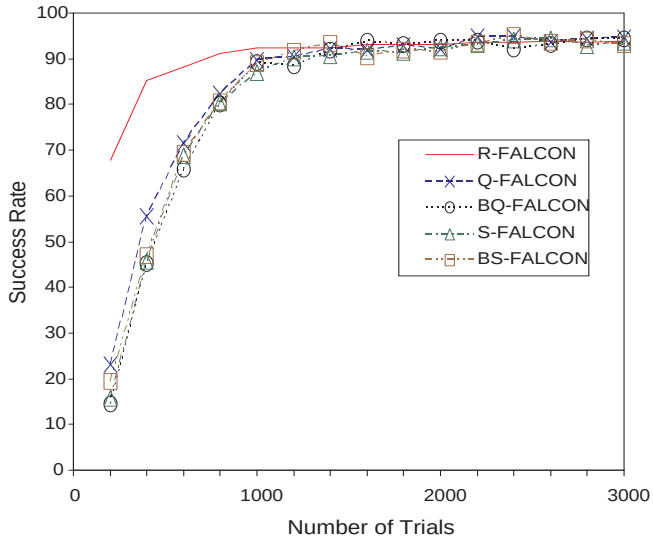


Figure 3.2: The success rates of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement over 3000 trials across ten experiments.

that the system has taken the optimal path to the target.

Figure 3.3 depicts the average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON to reach the target over 3000 trials across the ten sets of experiments. We see that all systems are able to reach the targets via near optimal paths after 1200 trials, although R-FALCON achieves that in 600 trials. In the long run, all systems produce a stable performance in terms of the quality of the paths taken.

Figure 3.4 depicts the average numbers of cognitive nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FLACON, and BS-FALCON over 3000 trials across the ten sets of experiments. Among the five systems, R-FALCON creates the most number of codes, significantly more than those created by the TD-FALCON systems. While we observe no significant performance difference among the four TD-FALCON systems in other aspects, BQ-FALCON and BS-FALCON demonstrate the advantage of the Bounded learning rule by producing a more compact set of cognitive nodes than Q-FALCON and S-FALCON.

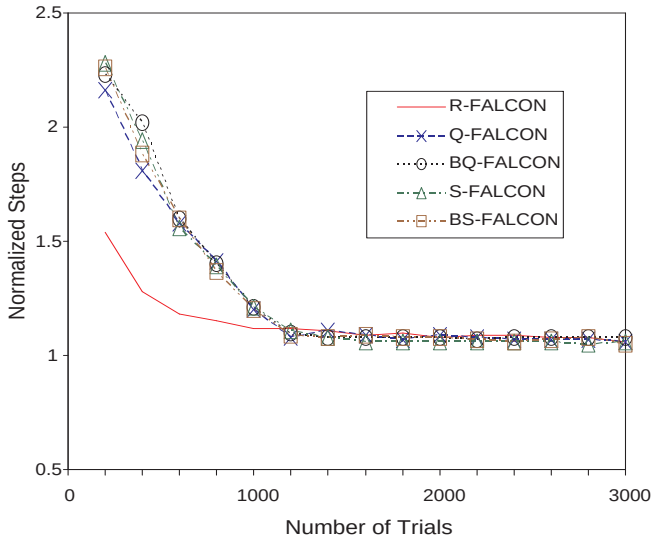


Figure 3.3: The average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement to reach the target over 3000 trials across ten experiments.

3.4.3 Learning with Delayed Reinforcement

In this set of the experiments, the AV does not receive immediate evaluative feedback for each action it performs. This is a more realistic scenario, because in the real world, the targets may be blocked or invisible. The reward scheme is described as follows: A reward of 1 is given when the AV reaches the target. A reward of 0 is given when the AV hits a mine. Different from the previous experiments with immediate rewards, a reward of 0 is given when the system runs out of time. In accordance with the Bounded Q-learning lemma, negative reinforcement values are not used in our reward scheme to ensure the Q-values are always bounded within the desired range of 0 and 1.

All systems use the same set of parameter values as shown in Table 3.2, except that the TD discount factor γ is set to 0.9 due to the absence of immediate reward signals. Figure 3.5 summaries the performance of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON in terms of success rates averaged at 200-trial intervals over 3000 trials across

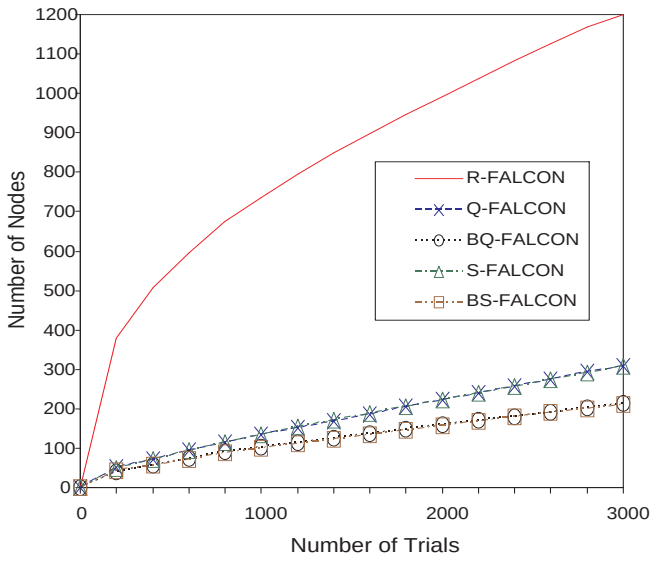


Figure 3.4: The average numbers of cognitive nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with immediate reinforcement over 3000 trials across ten experiments.

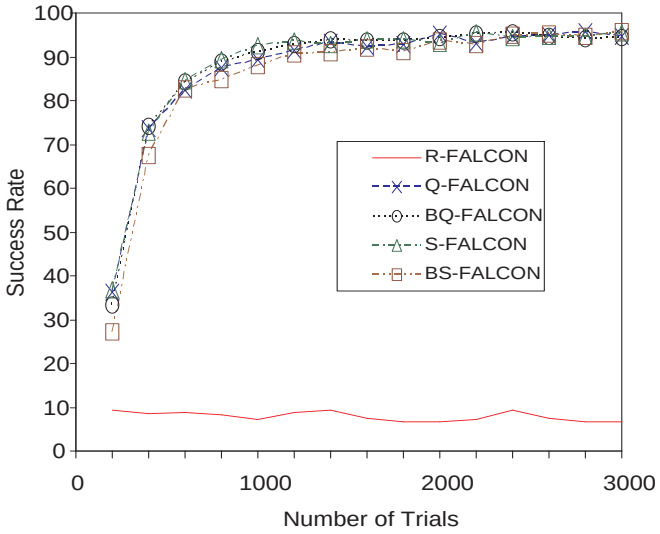


Figure 3.5: The success rates of R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement over 3000 trials across ten experiments.

ten sets of experiments. We see that R-FALCON produces a miserable near-zero success rate throughout the trials. This is not surprising as it only undergoes learning when it hits the target or a mine. The TD-FALCON systems, on the other hand, maintain the same level of learning efficiency as those obtained in the experiments with immediate reinforcement. At the end of 1000 trials, all four TD-FALCON systems can achieve success rates of more than 90%. In the long run, there is no significant difference in the success rates of the four systems.

Figure 3.6 shows the average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON to reach the targets over 3000 trials across ten experiments. Without immediate rewards, R-FALCON as expected performs very poorly. All four TD-FALCON systems, on the other hand, maintain the quality by always taking near optimal paths after 1000 trials.

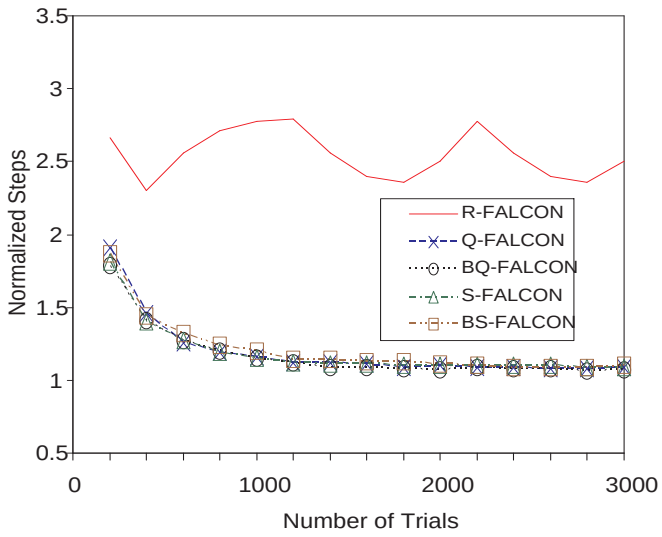


Figure 3.6: The average normalized steps taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement to reach the target over 3000 trials across ten experiments.

Figure 3.7 shows the numbers of cognitive nodes created by R-FALCON, Q-FALCON,

BQ-FALCON, S-FALCON, and BS-FALCON over the 3000 trials. Without immediate reward, the quality of the estimated value functions declines. As a result, all systems create a significantly larger number of cognitive nodes comparing with those created in the experiments with immediate reinforcement. Nevertheless, TD-FALCON systems with Bounded learning rule (i.e., BQ-FALCON and BS-FALCON) as before can cope with the delayed reinforcement using a smaller number of nodes.

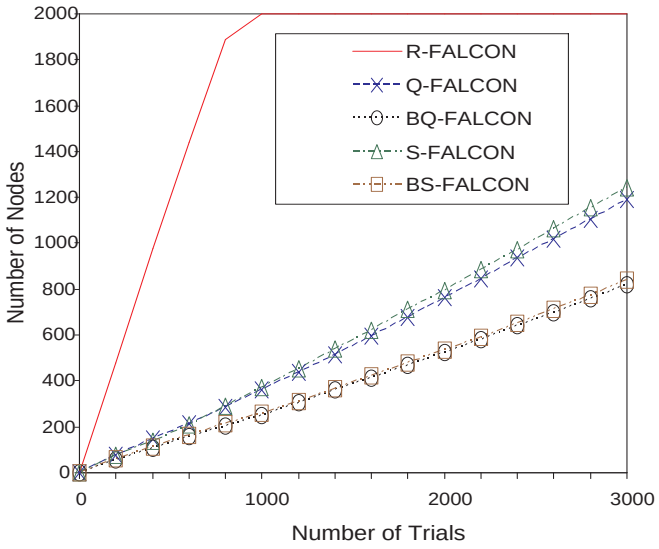


Figure 3.7: The average numbers of cognitive nodes created by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON operating with delayed reinforcement over 3000 trials across ten experiments.

3.4.4 Comparing with Gradient Descent based Q-Learning

To put the performance of TD-FALCON in perspective, we further conduct experiments to evaluate the performance of a reinforcement learning system (hereafter referred to as BP-Q Learner), using the standard Q-learning rule and a gradient descent based multi-layer feedforward neural network as the function approximator. Although we start off by incorporating temporal difference learning into the original (reactive) FALCON system for

the purpose of handling delayed rewards, FALCON effectively serves as a function approximator for learning the Q-value function. It thus makes sense to compare FALCON with another function approximator in the same context of Q-learning. Among the various universal function approximation techniques, we have chosen the gradient descent back-propagation (BP) algorithm as the reference point for comparison as it is by far one of the most widely used [78]. The specific configuration of combining Q-learning and multi-layer feedforward neural network with error backpropagation has been used by Sun *et. al* (2001) in a similar underwater minefield navigation domain.

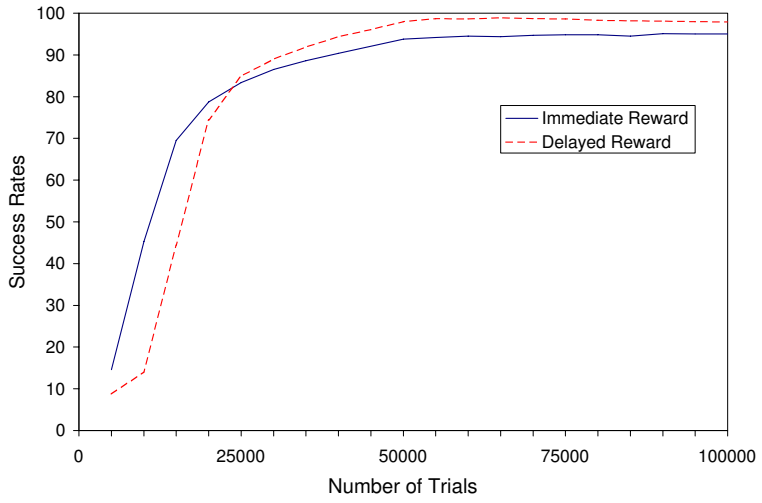


Figure 3.8: The success rates of BP-Q learning over $100,000$ trials across ten experiments.

The BP-Q Learner employs a standard three-layer (consisting of one input layer, one hidden layer, and one output layer) feed-forward architecture to learn the value function. The input layer consists of 18 nodes representing the five sonar signal values, eight possible target bearings, and five selectable actions. The input attributes are exactly the same as those used in the TD-FALCON, except that the sonar signals are not complement-coded. The output layer consists of only one node representing the value of performing an action in a particular state. All hidden and output nodes employ the symmetrical sigmoid function as Equation 2.5.

For a fair comparison, the BP-Q Learner also makes use of the same decayed ϵ-greedy

action selection policy. Using a learning rate of 0.25 and a momentum term of 0.5 for the hidden and output layers, we first experiment with a varying number of hidden nodes and obtain the best results with 36 nodes. Using a smaller number of, say 24, nodes produces a slightly lower success rate with a larger variance in performance. Increasing the number of nodes to 48 leads to a poorer result as well. We then experiment with different learning rates, from 0.1 to 0.3, for the hidden and output layers and obtain the best results with learning rates of 0.3 for the two layers. Increasing the learning rates to 0.4 and 0.5 produces slightly inferior results. We further experiment with different decay schedules for the ϵ-greedy action policy. We find that BP-Q requires a much longer exploration phase with an ϵ decay rate of 0.00001. Attempts with a higher ϵ decay rate meet with significantly poorer results. The best results obtained by the BP-Q Learner across ten sets of experiments in terms of success rates are reported in Figure 3.8. The performance figures, obtained with initial random weight values between -0.5 and 0.5, are significantly better than our previous results obtained using initial weight values between -0.25 and 0.25.

Although there has been no guarantee of convergence by using a function approximator, such as MLP with error backpropagation, for Q-learning [79], the performance and the stability of BQ-P are actually quite good. For both experiments involving immediate and delayed rewards, the BP-Q Learner can achieve very high success rates consistently, although it generally takes a large number of trials (around $40,000$ trials) to cross the 90% mark. In contrast, TD-FALCON achieves the same level of performance (90%) within the first 1000 trials. This indicates that TD-FALCON is around 40 times (more than an order of magnitude) faster than the BP-Q Learner in terms of learning efficiency.

Considering network complexity, the BP-Q Learner has the advantage of a highly compact network architecture. When trained properly, a BP network consisting of 36 hidden nodes can produce performance equivalent to that of a TD-FALCON model with around 200 cognitive nodes. In terms of adaptation speed, however, TD-FALCON is clearly a faster learner by successfully mastering the task in a much smaller number of trials.

3.5 Complexity Analysis

Subsequently we make an extensive analysis of space and time complexities, across the various TD-FALCON learners, including Q-FALCON, S-FALCON, R-FALCON, BQ-FALCON and BS-FALCON, as well as the traditional BP-Q Learner. We then put in comparison the computation times taken by the above systems in the minefield simulation experiments.

3.5.1 Space Complexity

The space complexity of FALCON is determined by the number of weight values or conditional links in the FALCON network. Specifically, the space complexity is given by $O((S+A+R)N)$, where S, A, and R are the dimensions of the sensory, action, and reward fields respectively, and N is the number of cognitive nodes in the category field. With a fixed number of hidden nodes, the space complexity of the BP-Q Learner is in the order of $O(S+A+R)$. BP-Q is thus typically more compact than a FALCON network.

Without function approximation, a table look-up reinforcement learning system [33, 45] would associate a value for each state or for each state-action pair. The space complexity for learning state-action mapping is thus $O(D^S)$, where S is the number of the sensory inputs and D is the largest number of discretized values across the S attributes. On the other hand, the space complexity for learning the state-action-value mapping is $O(D^S A)$, where A is the number of available actions. It can be seen that whereas the space complexities of TD-FALCON and BP-Q are in the order of polynomial, the space complexity of a traditional table look-up system is exponential.

3.5.2 Time Complexity

Table 3.3 summarizes the computational complexity of various FALCON systems compared with BP-Q, in terms of action selection and learning. For simplicity, we have omitted the dimension of reward field (R), which is fixed at 2. As TD-FALCON and BP-Q both compute the Q-values of all possible actions before selecting one, they have a higher time

	R-FALCON	Q-FALCON BQ-FALCON	S-FALCON BS-FALCON	BP-Q
Predicting	$O((S+A)N)$	$O((S+A)NA)$	$O((S+A)NA)$	$O((S+A)NA)$
Learning	$O((S+A)N)$	$O((S+A)NA)$	$O((S+A)N)$	$O((S+A)NA)$

Table 3.3: The time complexities of R-FALCON, TD-FALCON and BP-Q per sense-act-learn cycle. S and A denote the dimensions of the sensory and action fields respectively. N indicates the number of cognitive nodes for TD-FALCON and the number of hidden nodes in the context of BP-Q.

complexity than R-FALCON, which selects an action based on the current state input directly. In terms of learning, Q-FALCON, BQ-FALCON, and BP-Q are more time consuming as they need to evaluate the maximum Q-value of the next state. As TD-FALCON creates cognitive nodes dynamically whereas BP-Q uses a fixed number of hidden nodes, the latter one is deemed to have a lower time complexity. Based on the time complexity analysis, we conclude that the time costs of R-FALCON and BP-Q per reaction cycle are the lowest. Among the various TD-FALCON systems, the time complexities are basically equivalent with the size of the action space. The overall relations can be summarized as

$$\text{T(R-FALCON)} < \text{T(BP-Q)} <$$
$$\text{T(S-FALCON)} \equiv \text{T(BS-FALCON)} \equiv \text{T(Q-FALCON)} \equiv \text{T(BQ-FALCON)}$$

where $T(.)$ refers to the time complexity of the individual system, $<$ means "is lower than" and $\equiv$ means "is equivalent to".

3.5.3 Run Time Comparison

Table 3.4 and Table 3.5 show the computation times taken by the various systems per step (i.e., sense-act-learn cycle) in the minefield simulation experiments with immediate and delayed reinforcement respectively. The figures are based on our experiments conducted on a notebook computer using a 1.6GHz Pentium M processor with 512MB memory. For

experiments with immediate reinforcement, R-FALCON is the fastest by learning the action policy directly. BQ-FALCON and BS-FALCON are slower than R-FALCON, but are faster than S-FALCON and Q-FALCON. For experiments with delayed reinforcement, BQ-FALCON and BS-FALCON are also faster than Q-FALCON and S-FALCON. As the time complexities of the four TD-FALCON systems are in the same order of the magnitude, the variations in reaction time among the four TD-FALCON systems are largely due to the different numbers of cognitive nodes created by the various systems over the 3000 trials. On the whole, the reaction times per step for all systems are in the range of a few milliseconds. This shows that TD-FALCON systems are able to learn and function in real time with both immediate and delayed reinforcement.

Per experiment (3000 trials)	R-FALCON	Q-FALCON	BQ-FALCON	S-FALCON	BS-FALCON
Computing Time (seconds)	47.5	80.1	65.2	86.0	62.6
Average no. of Steps	25209	29224	30562	29416	28491
Computing Time/Step (ms)	1.9	2.7	2.1	2.9	2.2

Table 3.4: Computing times taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON and BS-FALCON for learning minefield navigation with immediate reinforcement.

Per experiment (3000 trials)	R-FALCON	Q-FALCON	BQ-FALCON	S-FALCON	BS-FALCON
Computing Time (seconds)	1741.6	241.4	166.2	255.5	172.9
Average no. of Steps	57892	29766	28646	27256	27969
Computing Time/Step (ms)	30.0	8.1	5.8	9.4	6.2

Table 3.5: Computing times taken by R-FALCON, Q-FALCON, BQ-FALCON, S-FALCON, and BS-FALCON for learning minefield navigation with delayed reinforcement.

Referring to Table 3.6, BP-Q tends to be more computationally expensive in the initial

learning stage. However, once the networks are fully trained, a minimal amount of time is spent in learning and the reaction time per cycle is extremely short. Averaged over 100,000 trials, the reaction time of BP-Q is 0.3 millisecond, even lower than those of TD-FALCON systems. However, BP-Q requires a much larger number of trials to achieve the same level of performance as TD-FALCON. The computing time required on the whole is in fact longer.

Per experiment (100,000 Trials)	BP-Q
Computing Time (seconds)	253.7
Average no. of Steps	975882
Computing Time/Step (ms)	0.3

Table 3.6: Computing time taken by BP-Q for learning minefield navigation. There is no noticeable difference between experiments with immediate and delayed reinforcement.

3.6 Summary

This chapter presents a neural model called FALCON, which is based on a multi-channel Adaptive Resonance Associative Map (ARAM) architecture, for multi-agent reinforcement learning. Operating in one of two modes, namely prediction and learning, using fuzzy ART operations, FALCON is able to perform online and incremental learning in a real-time environment.

TD-FALCON incorporates the Temporal Difference (TD) methods to estimate and learn value functions. During the iterative estimation of value function, the use of a temporal difference equation enables the TD-FALCON algorithm to be applied in multiple-step prediction problems. Using the Bounded Q-Learning rule, the Q-values can be self-scaled between 0 and 1, in the value function estimation.

Experiments in the minefield simulation environment with immediate reward show that the R-FALCON team creates significantly more cognitive nodes than TD-FALCON teams.

In the environment with only delayed reward, the R-FALCON team has a significantly poorer performance than TD-FALCON teams. The experimental results also show that TD-FALCON can learn much faster than the BP-Q learner.

In regard to space complexity, a traditional table look-up system is exponential. Both TD-FALCON and BP-Q are polynomial, but BP-Q is typically more compact than TD-FALCON network in space. In terms of the time costs, R-FALCON and BP-Q are the lowest, and various TD-FALCON learners are basically at the same level. From the computation times taken in the minefield simulation experiments, R-FALCON is the fastest learner, BQ-FALCON and BS-FALCON are slower than R-FALCON, but S-FALCON and Q-FALCON are the slowest. After the networks are fully trained, BP-Q has much less reaction time than those TD-FALCON systems.

Chapter 4

Cooperative Reinforcement Learning in Multi-Agent Domains

In typical multi-agent domains, the spatial relationship among agents is changing over time. We classify typical multi-agent domains into three categories. The first category consists of generic multi-agent domains, wherein the agents are tasked to fulfil their individual targets without cooperation, but have to avoid interfering with each other. TD-FALCON agents can adapt and function well in this category. For the second category, which includes the multi-agent domains that require simple cooperative mechanisms among agents, we propose the center of agent team (CAT) strategy to summarize the state space of all agents. For the third category, wherein complex multi-agent domains with a large number of input signals are included, we use an enhanced CAT strategy, namely the neighboring-agent mechanism (NAM), to further reduce the state space of each agent by blocking weak signals.

In this chapter, we use the multi-agent minefield navigation task, the multi-predator/prey pursuit game and the herding game as the benchmark problems, for the above-mentioned three categories of multi-agent domains respectively, to describe the mechanisms of the proposed cooperative learning approaches, and compare their performance with the state-of-the-art approaches.

4.1 Multi-Agent Minefield Navigation Task

In the multi-agent minefield navigation task, the game field, the settings and the game rules are almost the same to the minefield simulation task in Section 3.4.1. The difference is that in the multi-agent environment, there may be more than one autonomous vehicles (AVs) navigating through a minefield (Figure 4.1), requiring the agents to fulfil the task individually, but to avoid colliding with each other, which is deemed as the failure of the two colliding agents.

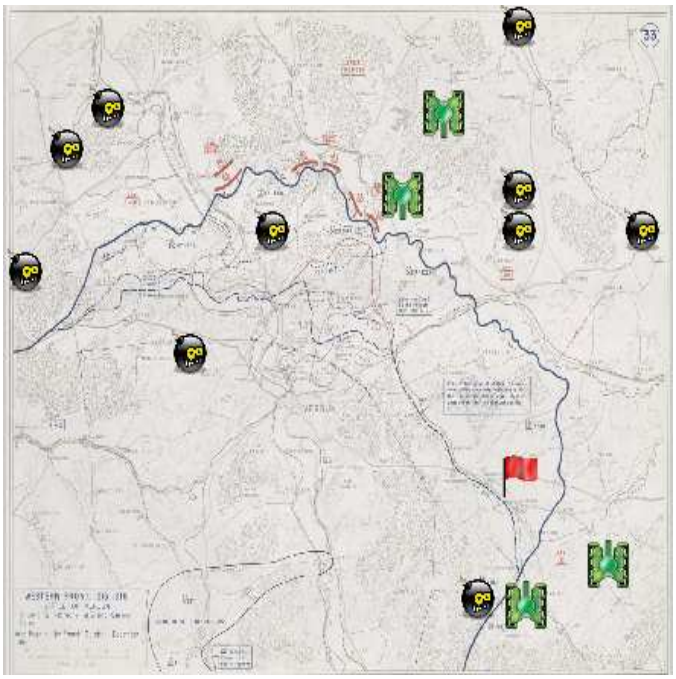


Figure 4.1: The multi-agent minefield navigation simulator.

4.1.1 State Representation and Reward Scheme

Comparing with the minefield simulation task, the multi-agent minefield navigation task becomes more challenging. Firstly, agents have to avoid collision with each other as it will result in the failures of both colliding agents. Secondly, as agents are non-stationary, an agent needs to predict the movement of its neighboring agents to avoid collision. For

the purpose of monitoring the other agents, an agent adds a separate set of five sonar sensors (using Equation 3.21) to its sensory representation, because agents and mines have different characteristics and thus should be tracked separately. Without the additional sonar sensors, our initial experiments produce poor results.

The complexity of learning in the multi-agent 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 $\mathbf{B} = 0, 1, \cdots, 7$ is the set of possible target bearings. With such a continuous state-action space, traditional reinforcement learning systems like the standard Q-learning, would have a scalability problem. Even if the sonar signals are converted to binary values, there are still at least $2^{10} * 5 * 8$ (approximately 40 thousand) possible combinations of state and action. On the contrary, based on the function approximation approach to Q-learning, the proposed TD-FALCON is able to compress the overall state-action space effectively, by using the template matching (Section 3.2.3) to control the expansion of the cognitive nodes.

We adopt the local reward mechanism in the multi-agent minefield navigation task, considering that there is no explicit cooperation among the agents. In detail, 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. At each step of the trial, an immediate reward is estimated by using Equation 3.22 from the minefield simulation task (Section 3.4.2).

4.1.2 Experimental Results

Subsequently we conduct three groups of comparative experiments: (1) performance of the TD-FALCON team in minefield environments with immediate and delayed reward respectively; (2) performance comparison between the TD-FALCON algorithm and the state-of-the-art approaches, including the backpropagation (BP) algorithm and the resilient propogation (RPROP) algorithm; (3) performance of the TD-FALCON team in multi-agent minefield navigation tasks under various scales.

In each experiment, all AVs are based on TD-FALCON with Bounded Q-learning (BQ-FALCON) denoted by Equation 3.14 and use the same set of the 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 temporal difference learning, the learning rate α is fixed at 0.5, the discount factor γ is set to 0.95, and the initial Q-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.

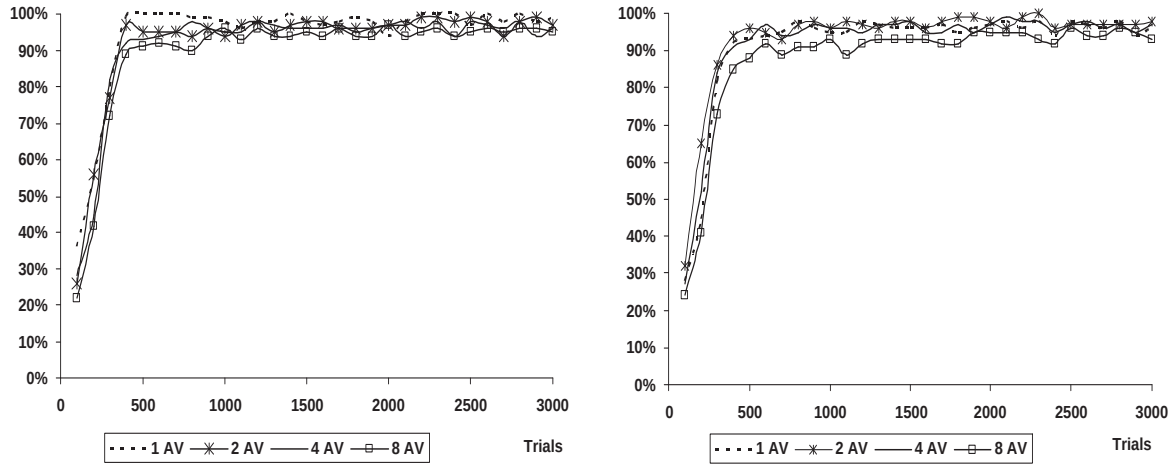
The settings of the experiments are basically same to the experiments in the minefield simulation task, as described in Section 3.4.1.

4.1.2.1 TD-FALCON Team with Immediate and/or Delayed Reward

In this group of experiments, we check the performance of a TD-FALCON team navigating in minefield domains with immediate and/or delayed reward, wherein each AV learns from scratch based on the feedback signals received from the environment. We run the experiments for 3,000 trials. In each trial, the initial locations of the AVs, the location of the target, and the locations of the mines are randomly set.

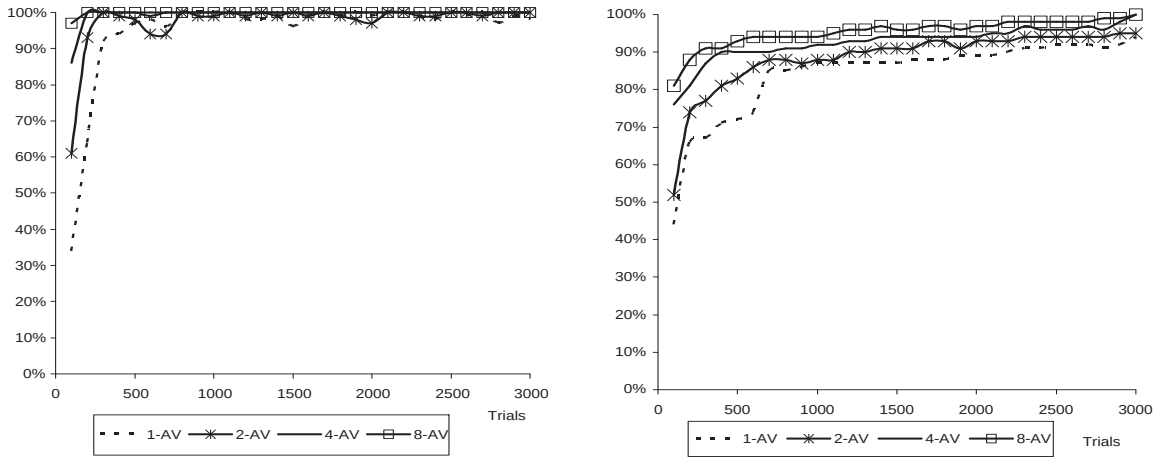
Figure 4.2 illustrates the performance of the TD-FALCON agent teams consisting of 1, 2, 4, and 8 AVs in terms of success rate averaged at 100-trial intervals within 3,000 trials. Here the success rate of a team in a trial is determined by the percentage of the AVs that reach the target position within the specified time in the trial. We can see that the success rate increases rapidly right from the beginning. By the end of 500 trials, almost all teams can achieve more than 90 percent success rate. Teams with fewer agents produce better results as there is a lower chance of collision.

To show the advantage of multi-agent TD-FALCON teams, we use the same group of TD-FALCON teams to repeat the experiments, redefining the success of an agent team as any one agent can reach the target. Figure 4.3 shows the performance of the TD-FALCON agent teams consisting of 1, 2, 4, and 8 AVs in terms of success rate averaged at 100-trial intervals within 3,000 trials, under the redefined success rate criteria. We can witness that



(a) Immediate reward (b) Delayed reward

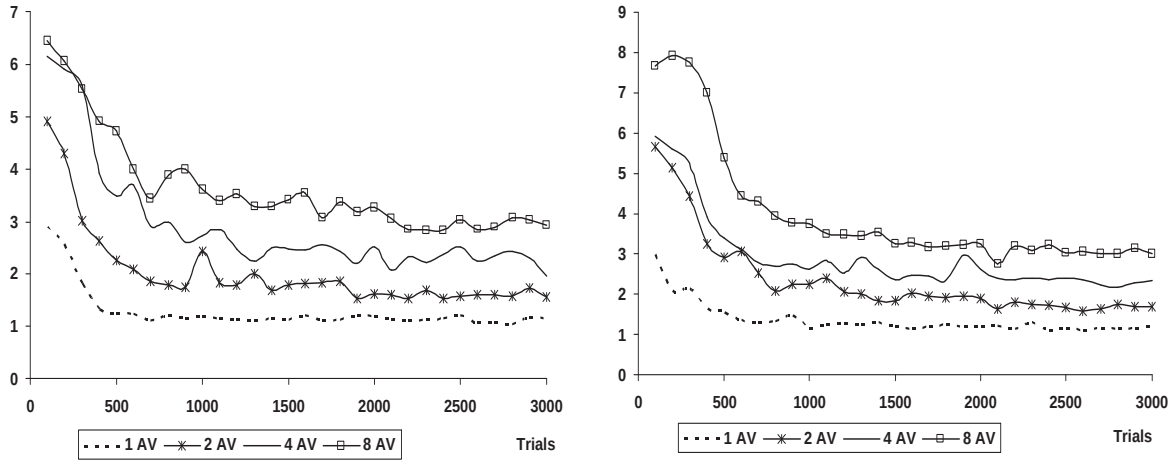
Figure 4.2: The success rates of TD-FALCON teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 100-trial intervals in a navigation task, based on the percentage of AVs reaching the target.



(a) Immediate reward (b) Delayed reward

Figure 4.3: The success rates of TD-FALCON teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 100-trial intervals in a navigation task, based on whether at least one agent reaches the target.

under the relaxed success criteria, teams with more agents produce a better performance, with either immediate or delayed rewards, as the possibility of at least one agent can reach the target has grown significantly with more agents involved. This trend is exactly the reverse of that in Figure 4.2.

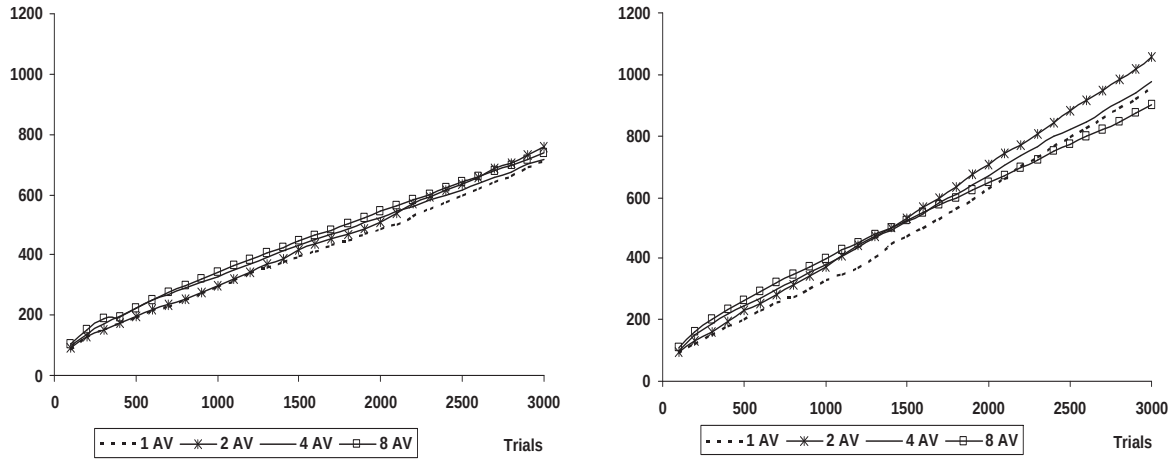


(a) Immediate reward (b) Delayed reward

Figure 4.4: The normalized steps of TD-FALCON teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 100-trial intervals in a navigation task.

Figure 4.4 depicts the normalized steps (measured by Equation 3.23) taken by the TD-FALCON agent teams consisting of 1, 2, 4 and 8 AVs averaged at 100-trial intervals. With just one agent, the normalized step converges to almost 1 after 1000 trials. Note that the normalized step values can seldom be 1's as detours are unavoidable when mines are on the optimal paths. Additionally, with more AVs in the system, the paths to the target are always less than optimal as the agents need to avoid collision with the mines as well as with each other.

Figure 4.5 illustrates the average numbers of cognitive nodes learned by each agent in the TD-FALCON agent teams consisting of 1, 2, 4 and 8 AVs averaged at 100-trial intervals. We observe that the average numbers of codes in all agent teams increase linearly over the learning trials. It is also noted that generally with more AVs involved, more cognitive nodes are developed. The reason is that there are more cases to be learned when more AVs



(a) Immediate reward

(b) Delayed reward

Figure 4.5: The numbers of cognitive nodes created by TD-FALCON teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 100-trial intervals in a navigation task.

work together.

It can be noticed that the TD-FALCON team with immediate reward outperforms the TD-FALCON team with delayed reward. Under the same number of agents, the success rate with immediate reward is 3∼5% higher than that with delayed reward on average. It is also noticed that the system with immediate reward has much less cognitive nodes learned than the system with delayed reward. After 3,000 trials, the teams with immediate reward generate only about 700 codes, but the teams with delayed reward generate about 1,000 codes. In terms of normalized steps, the system with immediate reward is a bit better than the system with delayed reward. After 3,000 trials, the 4-AV team with immediate reward achieves the normalized step of 1.95, but the 4-AV team with delayed reward can only achieve 2.34.

4.1.2.2 Comparison with the State-of-the-Art Reinforcement Learning Approaches

To put the performance of TD-FALCON in perspective, we further conduct comparative experiments with the traditional BP-Q Learner employed in Section 3.4.4. The perfor-

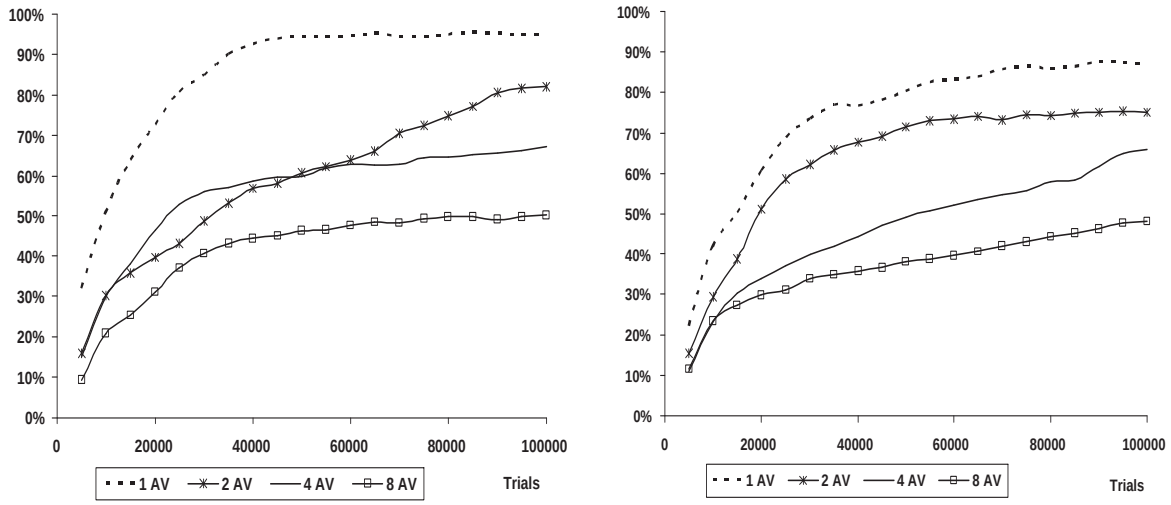
mances of BP-Q Learner teams in terms of success rate and normalized step over 100,000 trials are reported in Figure 4.6 and Figure 4.7 respectively.

Comparing Figure 4.6 with Figure 4.2, it can be found that the TD-FALCON learner evidently outperforms the BP-Q Learner in perspective of success rate. The 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams with immediate reward can achieve up to 95%, 82%, 67% and 50% success rates respectively, while the success rates of 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with immediate reward are 100%, 99%, 98% and 96% respectively. Moreover, the 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams with delayed reward can achieve up to 87%, 75%, 66% and 48% success rates respectively, while the success rates of 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with delayed reward are 98%, 97%, 95% and 95% respectively. It is also noticed that success rates of the BP-Q teams are significantly decreased with the growth of agent number, however, the TD-FALCON teams can adapt well and maintain the high level of success rates in multi-agent environments.

From the results of the comparative experiments, it can be found that the TD-FALCON agent teams learn much faster than the BP-Q reinforcement teams. For the 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams with immediate reward, it takes about 40000, 90000, 75000 and 60000 trials respectively to achieve peak values (95% of the highest values). In contrast, the 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with immediate reward can achieve peak values within only 400, 400, 500 and 500 trials respectively. Furthermore, for the 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams with delayed reward, it takes about 60000, 50000, 95000 and 90000 trials respectively to achieve peak values. On the contrary, the 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with delayed reward can achieve peak values within only 500, 500, 600 and 600 trials respectively. This indicates that the TD-FALCON learner is more than 100 times (over two orders of magnitude) faster than the BP-Q learner in terms of learning efficiency.

From Figure 4.7 and Figure 4.4, it is noticed that the TD-FALCON algorithm is also superior to the BP algorithm in respect to normalized step. After 100,000 trials, the 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams with immediate reward have the normalized steps of 1.30, 1.87, 2.12 and 3.15 respectively, and the 1-AV, 2-AV, 4-AV and 8-AV BP-Q teams

with delayed reward have the normalized steps of 1.38, 1.93, 2.33 and 3.46 respectively. On the contrary, after 3,000 trials, the 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with immediate reward have the normalized steps of 1.16, 1.56, 1.95 and 2.94 respectively, and the 1-AV, 2-AV, 4-AV and 8-AV TD-FALCON teams with delayed reward have the normalized steps of 1.18, 1.69, 2.34 and 2.99 respectively.



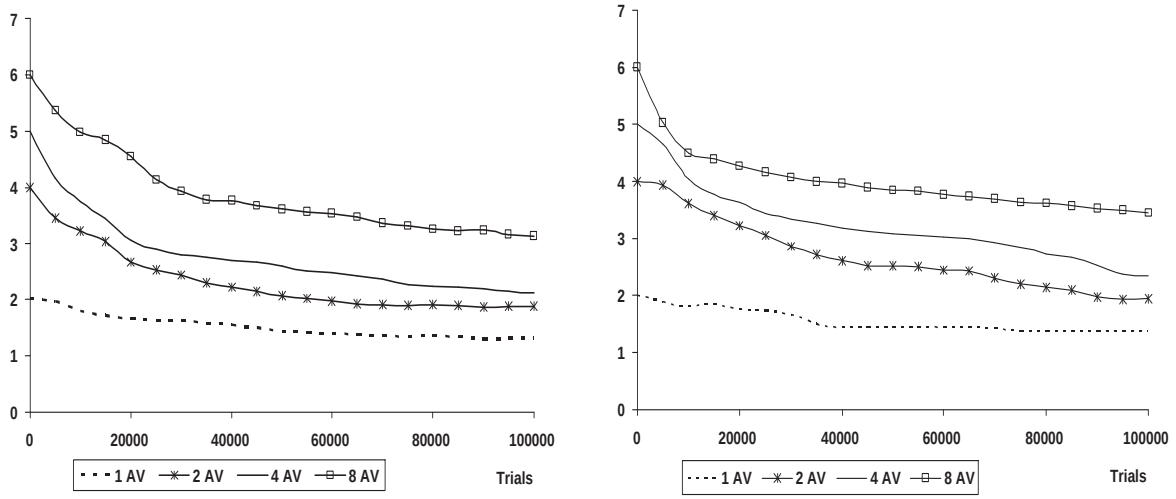
(a) Immediate reward (b) Delayed reward

Figure 4.6: The success rates of BP-Q teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 5000-trial intervals in a navigation task.

To further validate the performance of TD-FALCON, we repeat our experiments of the gradient descent based reinforcement learner, by replacing the standard BP algorithm with a faster learner, i.e., the resilient propogation (RPROP) algorithm [48, 80].

Similar to the BP-Q 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 a lot of empirical experiments and obtain the best performance of RPROP teams with 21 hidden units.

As shown in Figure 4.8, although RPROP shows a marked improvement over the BP algorithm, its success rate is still significantly lower than that of TD-FALCON. For example, under the immediate reward scheme, the 4-AV BP-Q team obtains 67% success rate after 100,000 trials, whereas the 4-AV RPROP team can achieve 74.2% success rate after 40,000



(a) Immediate reward

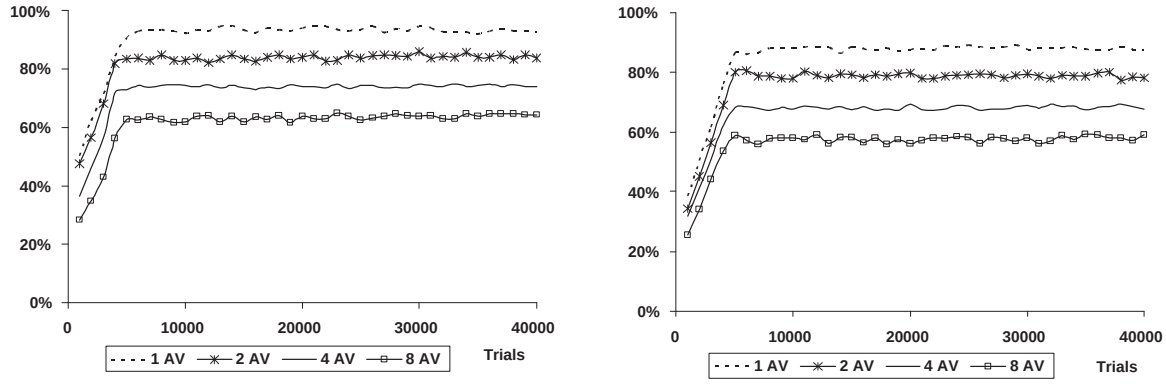
(b) Delayed reward

Figure 4.7: The normalized steps of BP-Q teams consisting of 1, 2, 4 and 8 agents with immediate and delayed reward averaged at 5000-trial intervals in the navigation task.

trials. However, the success rate of the 4-AV TD-FALCON team is 98% after only 3,000 trials. Similar to the BP-Q teams, the success rates of the RPROP teams also significantly drop from 92.8% with one agent to 64.3% with eight agents under the immediate reward scheme. In contrast, TD-FALCON can adapt well and maintain the high success rates in various multi-agent environments.

In terms of learning speed, RPROP indeed learns much faster than a BP-Q Learner. However, it still does not match the convergence speed of TD-FALCON. For example, under the immediate reward scheme, the 4-AV RPROP team achieves its peak performance within 5,000 trials. However, the 4-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, TD-FALCON is more than 100 times (two orders of magnitude) and 10 times (one order of magnitude) faster than the BP-Q Learner and the RPROP learner respectively.

To make a direct comparison, Figure 4.9 and Figure 4.10 collate the success rates of BP-Q, RPROP and TD-FALCON with one and eight agents respectively, operating under the same conditions for the first 3000 trials.



(a) Immediate reward

(b) Delayed reward

Figure 4.8: The success rates of RPROP teams averaged at 1000-trial intervals on the multi-agent minefield navigation task.

Referring to Figure 4.9, 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-Q and 71.4% of RPROP. In addition, the TD-FALCON learner also has a significantly lower normalized step of 1.158, compared with those of the BP-Q and RPROP learners. Experiments using the delayed reward scheme show similar results.

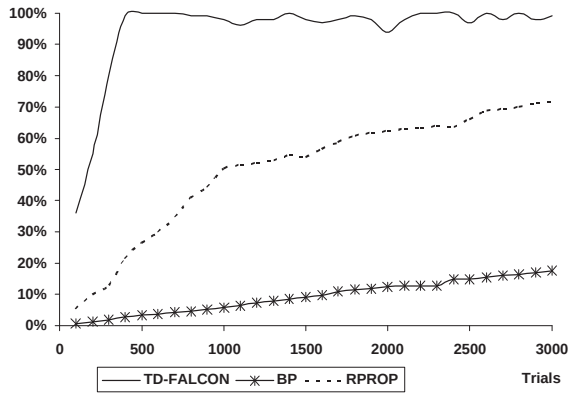
Referring to Figure 4.10, with immediate reward, the 8-AV TD-FALCON team has a much higher success rate (95.0%) after 3000 trials, compared with 5.7% of the BP-Q team and 43.0% of the RPROP team. On the average, the TD-FALCON team has a normalized step of 2.937, significantly lower than those of both the BP-Q and RPROP teams. Experiments using the delayed reward scheme again show similar results.

4.1.2.3 Scaling Up the Navigation Task

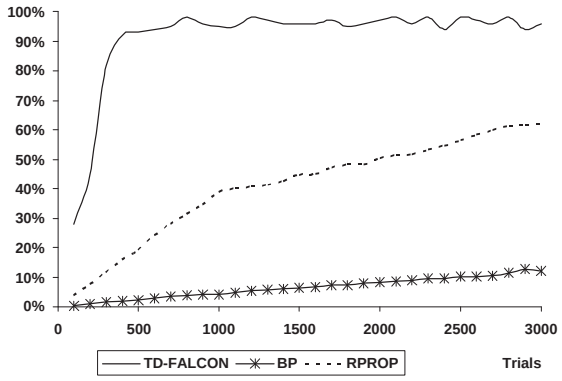
To demonstrate the scalability of the TD-FALCON algorithm, we further conduct experiments of TD-FALCON using a variety of minefield configurations as listed below.

1. 16 by 16 minefield with 10 mines, with the timeout threshold of 30 steps (original configuration)

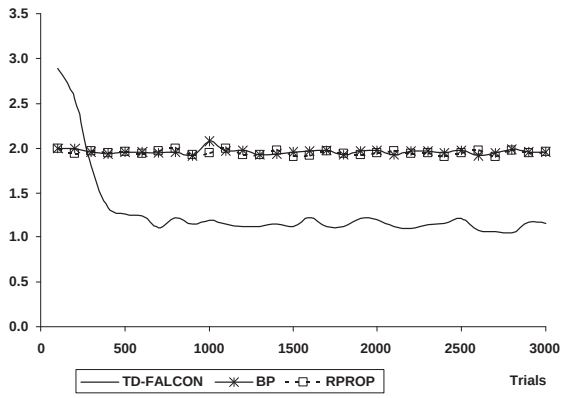
2. 32 by 32 minefield with 20 mines, with the timeout threshold of 60 steps



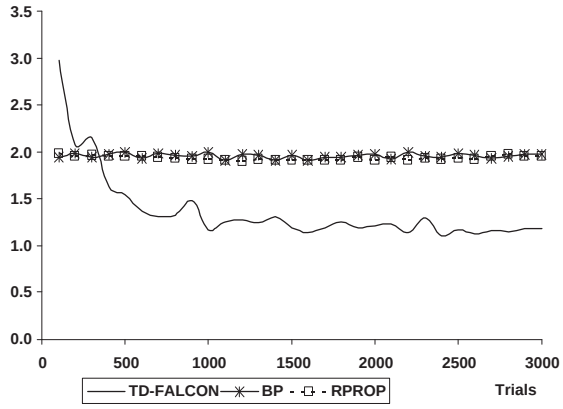
(a) Success rates with immediate reward



(b) Success rates with delayed reward

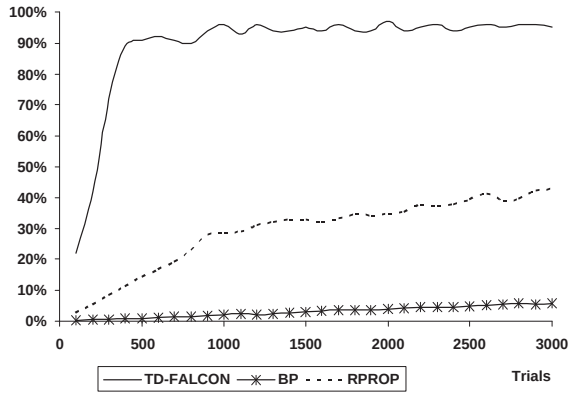


(c) Normalized steps with immediate reward

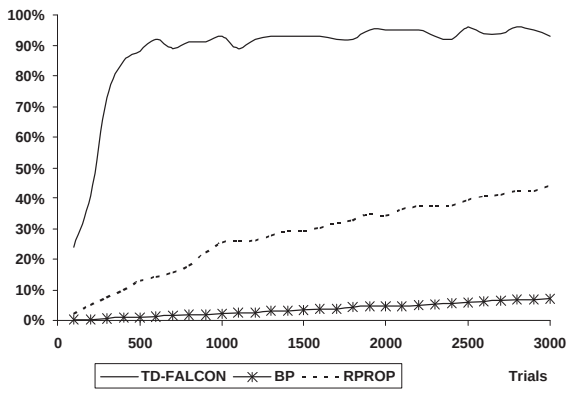


(d) Normalized steps with delayed reward

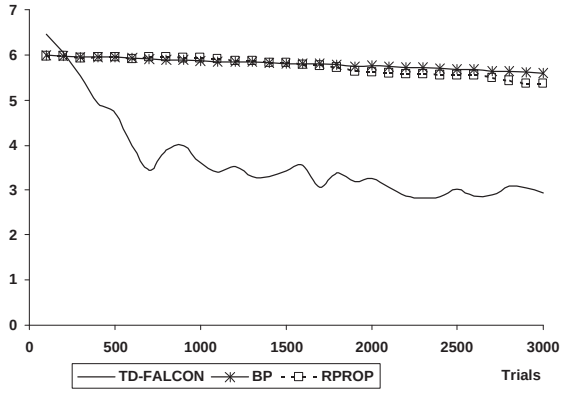
Figure 4.9: The success rates of TD-FALCON, BP-Q and RPROP teams with a single agent averaged at 100-trial intervals on the multi-agent minefield navigation task.



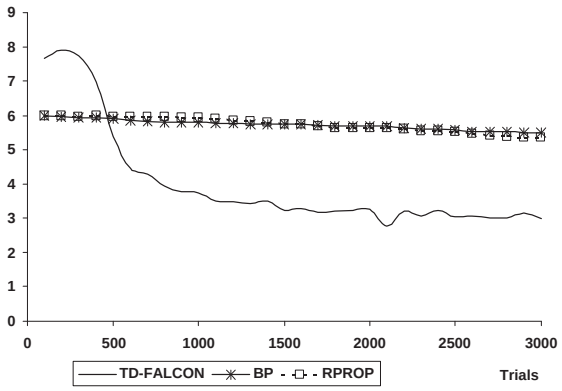
(a) Success rates with immediate reward



(b) Success rates with delayed reward



(c) Normalized steps with immediate reward

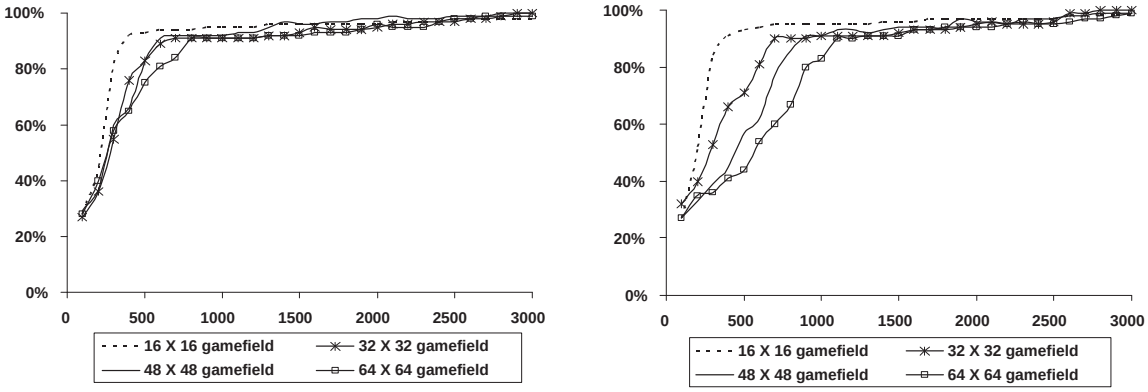


(d) Normalized steps with delayed reward

Figure 4.10: The success rates of TD-FALCON, BP-Q and RPROP teams with eight agents averaged at 100-trial intervals on the multi-agent minefield navigation task.

3. 48 by 48 minefield with 30 mines, with the timeout threshold of 90 steps

4. 64 by 64 minefield with 40 mines, with the timeout threshold of 120 steps



(a) Immediate reward (b) Delayed reward

Figure 4.11: The success rates of TD-FALCON teams consisting of four agents averaged at 100-trial intervals under various minefield configurations.

Figure 4.11 summarizes the success rates of TD-FALCON teams with four agents in the various minefield configurations. We see that TD-FALCON teams maintain a high success rate despite the increase in the domain size. With immediate reward, the success rates after 3000 trials are 98% and 95% in the smallest and the largest fields respectively. Similar results are observed for the delayed reward scheme.

Nevertheless, we note that the learning speed of a TD-FALCON team gets slower as the minefield size increases. With immediate reward, the TD-FALCON team with four agents achieves the peak performance at 500 trials in the smallest minefield but takes up to 1,000 trials in the largest one. The same observation is made for the delayed reward scheme. The increase in learning time should be attributed to the longer time needed to explore in a larger minefield.

4.2 The Predator/Prey Pursuit Game

As a typical multi-agent domain in the second category, the predator/prey pursuit domain [38, 81] has been widely used as a benchmark for multi-agent systems. Many variants of the pursuit domain exist with varied levels of complexity [82]. To create a challenging pursuit task, we use eight predators to capture a prey in a 16 by 16 field. The prey typically starts at the center of the field and the predators are distributed at the edges of the field (Figure 4.12). The rules of the game are summarized as follows.

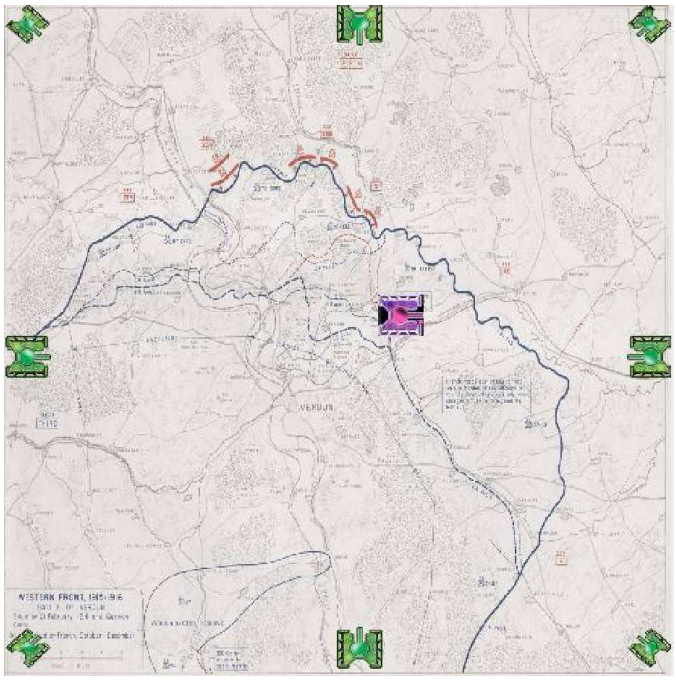


Figure 4.12: A layout of the pursuit game.

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 immovable. 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 [83], by which it selects an action that maximizes the total distance from the predators.

4. When any 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, the game is also considered as a failure.

4.2.1 The State Representation

For the purpose of cooperation, an agent needs to be aware of its surrounding agents. Under traditional cooperative strategies, the state space can be very large in the pursuit game. For example, using the monolithic Q-learning [18, 17, 84], wherein the state space of each agent is increased with the information of the other agents, 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 [56]. 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 $(c_1, c_2, \cdots, c_n)$, 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 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 $S = N^8 = 961^8 = 727,423,121,747,185,263,828,481$. In consideration of the issue, we should work out cooperative strategies to effectively reduce the state space in the predator/prey pursuit game.

4.2.1.1 All-Bearing Strategy

Similar to the minefield navigation problem, our formulation of the predator/prey pursuit game also follows the Partially Observable Markov Decision Process (POMDP). Based on our analysis of the problem, the optimal action of a predator depends on its bearing rather than its distance to the prey. This implies that the canonical distance between a predator 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

$$B^i_j = \begin{cases} 1 & \text{if } j = b^i \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

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 a eight by eight state space.

4.2.1.2 CAT Bearing Strategy

Instead of representing the bearing of each agent, we introduce a high level concept called *Center of Agent Team* (CAT) and incorporate into the state vector the bearing from the CAT to the prey. Assume that each of the eight predators P_i has the coordinates (x_i, y_i) $(1 \leq i \leq 8)$, then the CAT P_c refers to the point with the coordinates (x_c, y_c), where

$$x_c = \sum_{i=1}^{8} \frac{x_i}{8}, y_c = \sum_{i=1}^{8} \frac{y_i}{8}. \tag{4.2}$$

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

$$B^c_j = \begin{cases} 1 & \text{if } j = b^c \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

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 control the state space, we adopt binary sonar inputs to detect the existence of predators in the immediate surrounding. The predator sonar vector $\mathbf{S}^i = (a_0, a_1, \ldots, a_{15})$ $(1 \leq i \leq 8)$ is given by

$$a_j = \begin{cases} 1 & \text{if } j < 8 \text{ and } d_j = 1 \\ 0 & \text{if } j < 8 \text{ and } d_j > 1 \\ 1 - a_{j-8} & \text{if } j \geq 8. \end{cases} \tag{4.4}$$

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.

4.2.2 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 [85] 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 *Center of Agent Team* (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 $r_i = \frac{1}{d^i(1+d^c)}$, 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, with $d^i = 1$ and $d^c = 0$, the above reward function produces a value of 1. However, when two predators collide, a final reward signal of 0 is given. Likewise, when the prey reaches the edge of the game field, a reward

of 0 is provided to all the agents.

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

Table 4.1: The state vectors and reward functions used by the various cooperative strategies.

4.2.3 Experimental Results

Now we arrange two groups of comparative experiments: (1) performance comparison among the team of non-cooperating agents, the team using the All-Bearing strategy and the team using the CAT-Bearing strategy; (2) performance comparison between the TD-FALCON algorithm and the resilient propogation (RPROP) algorithm, using the same CAT-Bearing cooperative strategy. The game field of the experiments is shown in Figure 4.12, and the configuration of the testing environment is same as Section 4.2.

The testing environment is illustrated in Figure 4.12. TD-FALCON teams in all experiments use the following parameter values: choice parameter $\alpha^{ck} = 0.001$, learning rate $\beta^{ck} = 1.0$ (fast learning), baseline vigilance parameter $\bar{\rho}^{ck} = 0.9$ for $k = 1$ to 3, initial exploration rate $\epsilon = 0.6$, decayed rate $d_\epsilon = 0.0004$, Q-learning rule learning rate $\alpha = 0.5$, and discount parameter $\gamma = 0.01$.

4.2.3.1 Performance Comparison among Various Cooperative Strategies

In this group of tests, we conduct three sets of comparative experiments to evaluate the efficacy of the various cooperative strategies. The first set of the experiments, involving teams of non-cooperating agents with relative positional information in the state representation, provides the baseline performance for comparison. The relative positional information is denoted by $\mathbf{D} = (s_0, s_1)$, where $\mathbf{s_0} = \frac{1}{d}$, $s_1 = 1 - s_0$, and d is the distance from this agent to the prey. The second set of the experiments involves TD-FALCON teams using the All-Bearing cooperative strategy. The last set of the experiments employs 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 4.1.

Figure 4.13 shows that cooperating TD-FALCON teams have a clear performance advantage over their non-cooperating counterpart. After $6,500$ trials, the success rates of cooperating teams reach 95%, compared with less than 85% success rate achieved from the non-cooperating team.

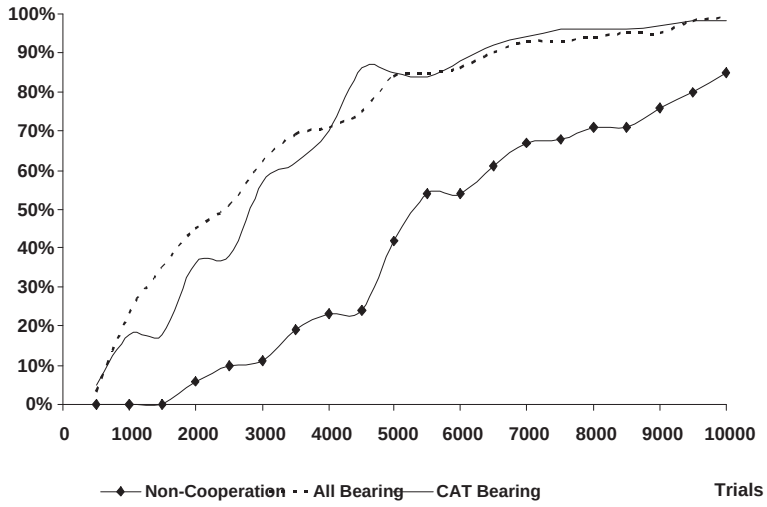


Figure 4.13: The success rates of TD-FALCON teams using the three strategies on the pursuit game.

Figure 4.14 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 non-cooperating team.

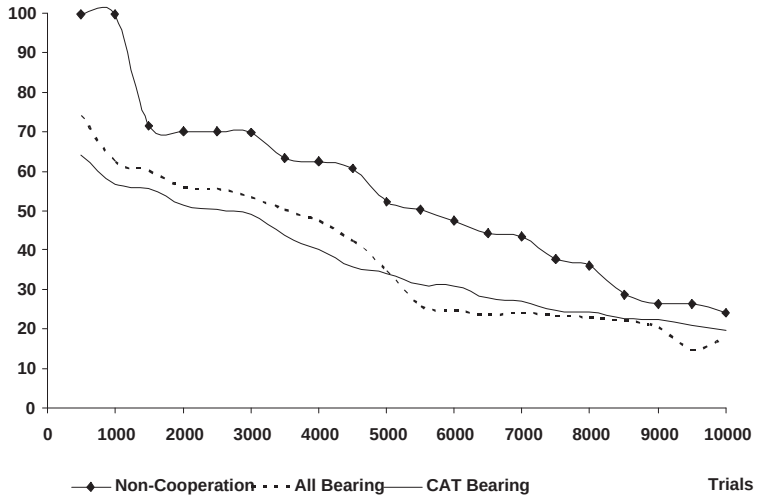


Figure 4.14: The numbers of steps taken by TD-FALCON teams using the three strategies on the pursuit domain.

Figure 4.15 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.

4.2.3.2 Performance Comparison between TD-FALCON and RPROP Algorithms

To put the performance of TD-FALCON in perspective, we repeat the experiments using the same settings as Section 4.2.3.1, with the RPROP reinforcement learner. Figure 4.16 shows the success rate 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 TD-FALCON. After 10,000

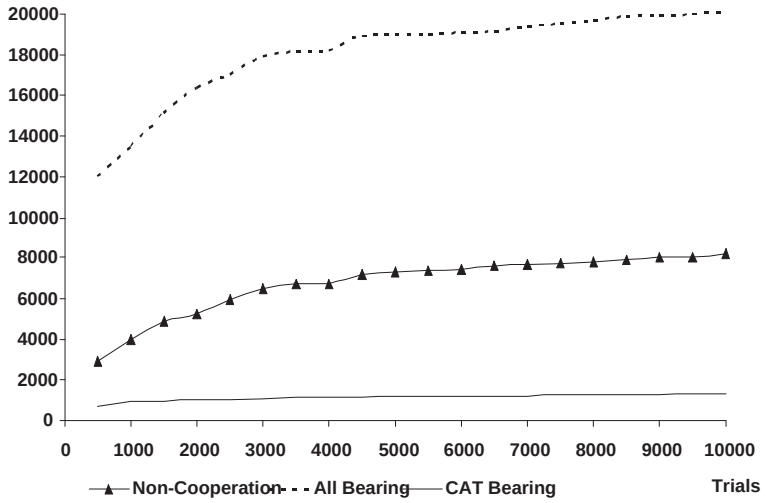


Figure 4.15: The numbers of cognitive nodes learned by TD-FALCON teams using the three strategies on the pursuit domain.

trials, the success rate of the TD-FALCON team is typically above 95%. However, the RPROP team can only achieve less than 50% success rate.

Figure 4.17 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 much 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.

4.3 Herding Game

As a typical multi-agent domain of the third category, the herding game requires three shepherds to encircle and force a sheep to enter the corral, which is at a boundary of a grassland. Initially, the shepherds are located at three boundaries respectively and the sheep is located at the center of the grassland (Figure 4.18(a)). The game rules of the herding task are: (1) At each time step, a shepherd can move in the direction of left, right,

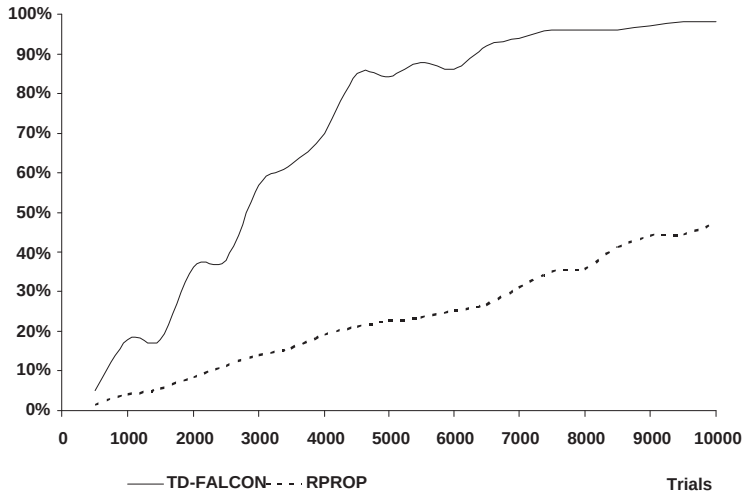


Figure 4.16: The success rates of TD-FALCON and RPROP teams using the CAT bearing strategy on the pursuit domain.

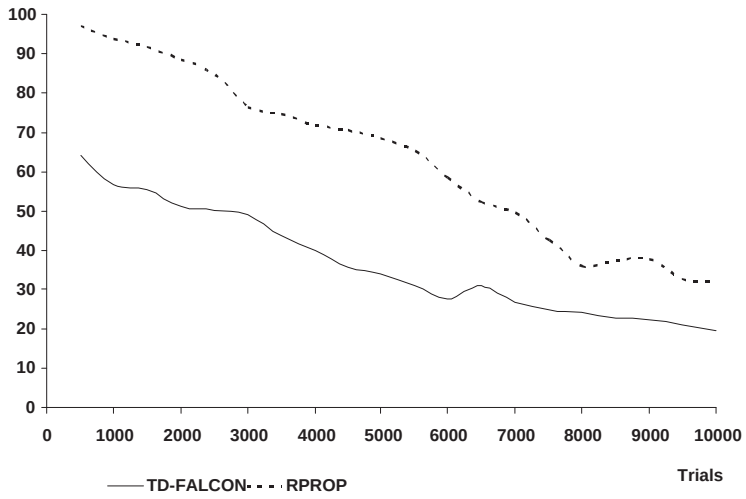
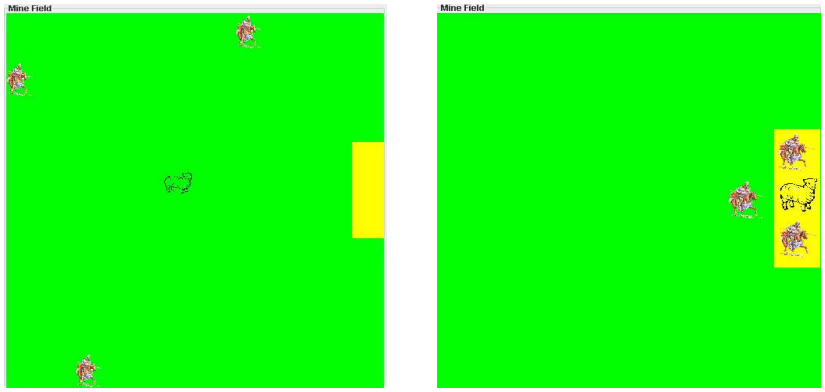


Figure 4.17: The numbers of steps taken by TD-FALCON and RPROP teams using the CAT bearing strategy on the pursuit domain.

up or down with the pace of one square, or just stay stationary. (2) The sheep moves one square every two time steps. (3) The sheep tries its best to escape to the boundary with the shortest distance. However, if there is only one way available for the sheep, it must go that way. (4) If the sheep is forced to enter the corral, the game is ended successfully (Figure 4.18(b)). If the sheep manages to escape to any boundary, or if the game is unfinished within a time limit, the game fails.



(a) Starting scenario (b) Success scenario

Figure 4.18: Starting and Success Scenarios in the herding task.

Obviously the herding game is more complex than the predator/prey pursuit task, for the agents (shepherds) in the herding game need to not only half-encircle the target (sheep), but also control the moving direction and the pace of the target. With the growth of the complexity, the TD-FALCON system needs to develop more cognitive nodes to deal with the increased cases. Therefore the scalability issue may be caused.

4.3.1 Neighboring-Agent Mechanism

Addressing the increased complexity of the herding game and based on the CAT algorithm, we have developed a new cooperative strategy, namely neighboring-agent mechanism (NAM), to make the TD-FALCON system more flexible and adaptable. According to the NAM algorithm, only signals from neighboring (detectable) agents are received and processed. The purpose of using the NAM algorithm in multi-agent environment is to further compress the state space.

Assume that in a multi-agent game field, there are n moving agents in a NAM team, marked as $A_1, A_2, \cdots, A_n$, and a moving target, marked as T. Let $d^{ij}(1 \leq i, j \leq n)$ be the distance between A_i and A_j. Let $\overline{d}^i$ $(1 \leq i \leq n)$ be the distance between A_i and T. Let $b^{ij}(1 \leq i, j \leq n)$ be the bearing from A_i to A_j. Let $\overline{b}^i(1 \leq i \leq n)$ be the bearing from A_i to T. Let σ be the Critical-Point Distance (CPD), which is used as a criterion to filter input signals in the NAM algorithm.

Definition: If $d^{ij} \geq \sigma(1 \leq i, j \leq n)$ or $\overline{d}^i \geq \sigma(1 \leq i \leq n)$, we say that A_j is **undetectable** from A_i or T is **undetectable** from A_i. Otherwise, If $d^{ij} < \sigma(1 \leq i, j \leq n)$ or $\overline{d}^i < \sigma(1 \leq i \leq n)$, we say that A_j is **detectable** to A_i or T is **detectable** to A_i.

The principles of the NAM algorithm are: (1) A_i can not receive any signal from A_j until $d^{ij} < \sigma$. (2) A_i can not receive any signal (except the bearing signal) from T until $\overline{d}^i < \sigma$. (3) To reduce the state space of A_i, the signals (if receivable) from multiple other agents are summarized as if they were from a "virtual" agent A_I, which is the centre of those detectable agents, before being processed.

In general, the sensory inputs of an agent A_i $(1 \leq i \leq n)$ can be denoted by $S^i = (S^{it}, S^{io})$, where S^{it} and S^{io} denote sensory inputs from T and A_I respectively.

The sensory inputs that the agent A_i receives from the target T can be further denoted by $S^{it} = (D^{it}, B^{it})$, where D^{it} stands for the distance signal that A_i receives from T, and B^{it} stands for the bearing vector that A_i receives from T. Furthermore, the distance signal that A_i receives from T is computed by

$$D^{it} = \begin{cases} \frac{1}{\overline{d}^i} & \text{if } \overline{d}^i < \sigma \\ \varepsilon & \text{if } \overline{d}^i \geq \sigma \end{cases} \tag{4.5}$$

where ε is a dummy value, used for keeping the dimension of sensory inputs fixed. To compute the bearing vector B^{it}, we define eight directions, hence there is $0 \leq \overline{b}^i \leq 7$, and the bearing vector B^{it} is

$$B^{it}_k = \begin{cases} 1 & \text{if } k = \overline{b}^i \\ 0 & \text{otherwise,} \end{cases} \tag{4.6}$$

where $0 \leq k \leq 7$.

The sensory inputs that A_i receives from A_I is $S^{io} = (D^{io}, B^{io})$, where D^{io} is the distance signal, and B^{io} is the bearing vector. The distance signal is

$$D^{io} = \begin{cases} \frac{1}{d^{io}} & \text{if any other agents detectable} \\ \varepsilon & \text{otherwise} \end{cases} \tag{4.7}$$

where d^{io} is the distance between A_i and A_I. The computation of the bearing vector B^{io} is a bit more complex. We consider the problem in two conditions. If there are other agents detectable to the agent A_i, we use the following equation:

$$B_k^{io} = \begin{cases} 1 & \text{if } k = b^{io} \\ 0 & \text{otherwise,} \end{cases} \tag{4.8}$$

where $0 \leq k \leq 8$, and b^{io} $(0 \leq b^{io} \leq 8)$ is the bearing from A_I to A_i. This bearing vector has an extra value denoting the case of the coincidence of A_I and A_i. If no other agents are detectable, the bearing vector B^{io} is

$$B_k^{io} = 0. \tag{4.9}$$

To obtain the bearing and the distance from A_I to A_i, we should know the coordinates (relative to A_i) of A_I. Assume that among the $n-1$ agents, there are p $(1 \leq p \leq n-1)$ agents, marked as $A_1^O, A_2^O, \cdots, A_p^O$, detectable to the agent A_i. The coordinates (relative to A_i) of an agent A_m^O $(1 \leq m \leq p)$ can be obtained as (x^m, y^m) $(-\sigma < x^m, y^m < \sigma)$, then the coordinates (x^o, y^o) (relative to A_i) of A_I are

$$x^o = \sum_{m=1}^{p} \frac{x^m}{p}, y^o = \sum_{m=1}^{p} \frac{y^m}{p}. \tag{4.10}$$

Then the distance d^{io} is

$$d^{io} = max(|x^o|, |y^o|). \tag{4.11}$$

The state space of A_i can be computed as

$$|S^i| = |D^{it}| \cdot |B^{it}| \cdot |D^{io}| \cdot |B^{io}|. \tag{4.12}$$

Furthermore, $|D^{it}| = \sigma + 1$, $|B^{it}| = 8$, $|D^{io}| = \sigma + 1$ and $|B^{io}| = 9$. The overall state space of A_i is

$$|S^i| = 72(\sigma + 1)^2. \tag{4.13}$$

Since σ is a constant, the state space of an agent in the NAM algorithm is only $O(1)$.

Similar to the CAT algorithm [31], we use a hybrid reward function incorporating individual and team payoffs in the NAM algorithm. The reward function of an agent A_i is denoted by $r_i = \frac{1}{d^i(1+d^c)}$, where d^i and d^c are the distances from A_i and the *Center of Agent Team* (CAT) to T respectively.

4.3.2 Experimental Results

The experiments on the herding task are performed on a 16 by 16 game field. We would evaluate the performance of the NAM team, in comparison with the CAT and the traditional joint action mechanism (JAM) [33, 34, 35] teams. For the NAM team, we take $\sigma = 2$ and $\varepsilon = 0.001$. The testing environment is shown in Figure 4.18.

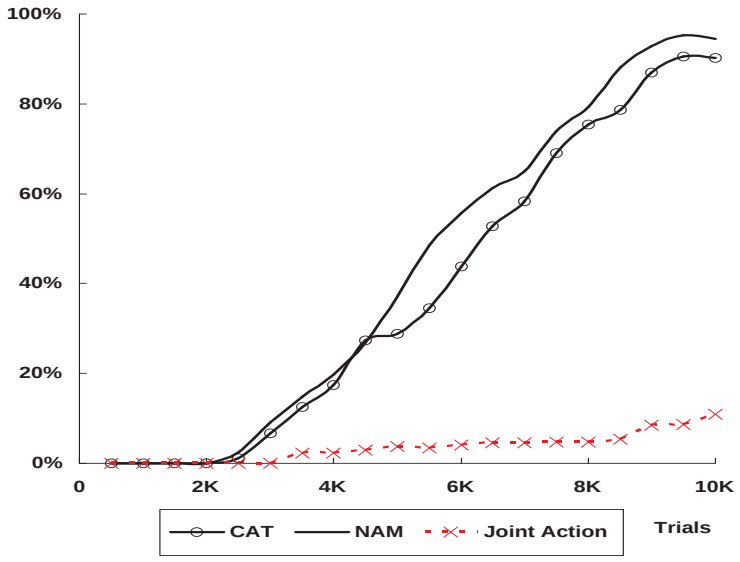


Figure 4.19: The success rates of the TD-FALCON teams with CAT, NAM and joint action algorithms in the herding game.

Figure 4.19 highlights the success rates of TD-FALCON teams implemented with the CAT, the NAM and the joint action algorithms respectively, averaged at 500-trial intervals across 10,000 trials, in the herding task. It is witnessed that both of the CAT and NAM

teams can adapt and function well in the herding game. However, the NAM team can produce even better performance than the CAT team. After 10,000 trials, the agent team implemented with the CAT algorithm can achieve about 90% success rate, but the agent team implemented with the NAM algorithm can achieve more than 95% success rate. Figure 4.19 also shows that the TD-FALCON team implemented with the joint action mechanism converges very slowly. Even after 10,000 trials, the team can only achieve 10.9% success rate.

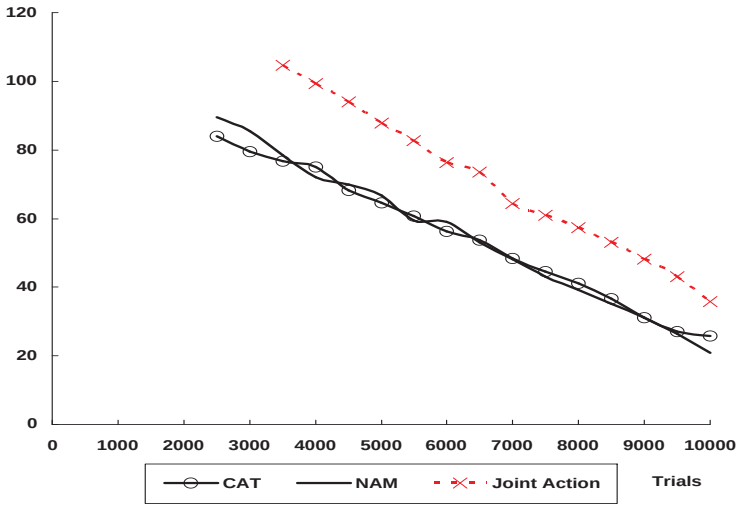


Figure 4.20: The numbers of steps of the TD-FALCON teams with CAT , NAM and joint action algorithms in the herding game.

Figure 4.20 shows the numbers of steps that the three individual agent teams take to encircle and force the sheep to enter the corral, averaged at 500-trial intervals across 10,000 trials. It is found that the joint action team takes much more steps than the other two teams. After 10,000 trials, the joint action team has the average number of 35.8 steps. It is also noticed that the NAM team is able to produce a bit fewer steps than the CAT team. After 10,000 trials, the CAT team has the average number of 25.8 steps, but the NAM team only has the average number of 20.8 steps.

Figure 4.21 shows the numbers of cognitive nodes generated from the three agent teams

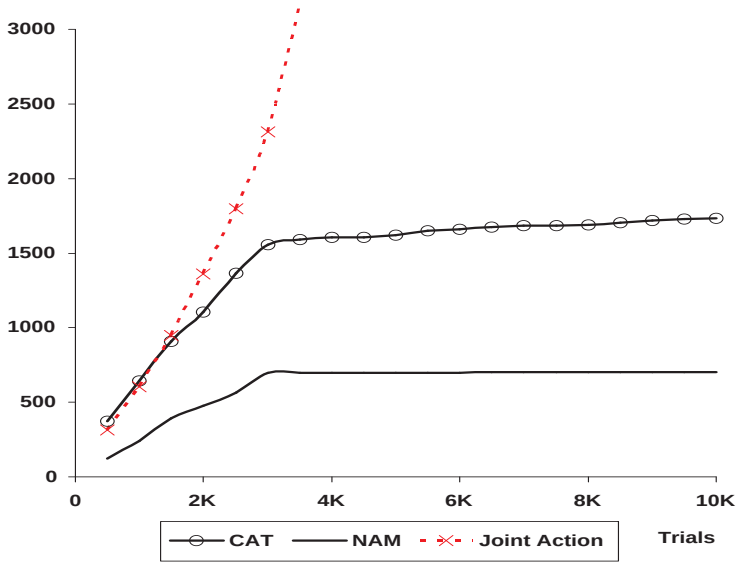


Figure 4.21: The numbers of cognitive nodes of the TD-FALCON teams with CAT , NAM and joint action algorithms in the herding game.

respectively, averaged at 500-trial intervals across 10,000 trials. The number of nodes in the joint action team is increased very rapidly. After 3,000 trials, it has produced 3,152 nodes. It is obvious that the NAM team produces much fewer cognitive nodes than the CAT team. After 10,000 trials, the NAM team produces 701 cognitive nodes, in contrast to 1,735 cognitive nodes created from the CAT team.

4.4 Summary

In this chapter, the cooperative reinforcement learning is studied in the environment of typical multi-agent domains.

Firstly, we use the multi-agent minefield navigation task as a domain of study, wherein agents are required to fulfill their own tasks without cooperation, but need to avoid conflicting with each other. The experimental results show that (1) TD-FALCON agents can adapt and function well in the minefield navigation task; (2) TD-FALCON teams can significantly outperform the BP-Q teams; (3) TD-FALCON teams can maintain a high level

performance with the domain size scaling up.

Secondly, we study the multi-predator/prey pursuit game, which is challenging due to the huge state space. We propose the Center of Agent Team (CAT) strategy to reduce the state space significantly. The CAT strategy has two main points: (1) Use bearing in place of distance to represent the state space; (2) Use CAT to summarize the state space of all other predators. In regard to the reward scheme, we adopt the combination of local and global rewards. The experimental results demonstrate that the CAT strategy can prominently outperform the other cooperative strategies. From the experiments, it is also noticed that the TD-FALCON team has a much better performance than the RPROP team.

Finally, we study the herding game as a benchmark problem, which is a very tough task, because the agents need to not only half-encircle the moving target, but also control the moving direction and the pace of the target. Based on the CAT algorithm, we have developed the neighboring-agent mechanism (NAM) to further compress the state space and make the TD-FALCON system more flexible and adaptable. The key point of the NAM algorithm is that only signals from neighboring agents are received and processed. The experimental results indicate that the NAM algorithm can achieve a higher performance than the CAT algorithm.

Chapter 5

Cooperative Reinforcement Learning in Topology-based Multi-Agent Systems

Traditional multi-agent systems [3, 9] assume that all agents can interact freely with each other and do not consider specific relations among the agents. Now we study a special group of multi-agent domains, namely topology-based multi-agent systems (TMAS), wherein each individual agent interacts only with its neighbors according to their spatial relationship. TMAS systems are well suited for problems with topological constraints, of which network routing (NR) is a classical example. Other TMAS domains include distributed vehicle monitoring [86, 87], network management and routing [20, 22], electricity distribution management [21, 88] etc.

Consider the network routing (NR) problem, wherein the objective is to send a number of packets between two points through a network [89]. A direct solution to the NR problem is the so-called *single agent strategy*, which employs one autonomous agent for each routing node, and each agent learns an optimal policy through the sensory and reinforcement feedback from their neighbors and the environment. However, in a TMAS system, the single agent strategy may be encountered with many challenges. First of all, the agents

have to handle the added spatial constraints. With a unique topological relationship with its neighbors, each agent has to deal with a different set of state space. In addition, as the number of agents increases, the agents have to adapt to the increased complexity of the spatial relationship.

Facing those challenges from TMAS domains, we propose a TD-FALCON Binary Tree Formation (BTF) strategy, wherein a team of TD-FALCON agents, organized in binary tree formation, is deployed for each routing node. As each TD-FALCON agent in the BTF strategy now interacts with its neighboring TD-FALCON agents in a fixed (binary) topology, the state space does not increase with the complexity of the network topology. In addition, the uniform state space representation also enables the agents to perform knowledge sharing [18, 90, 91], thus boosting the learning and operational efficiency.

TD-FALCON agents can also be deployed at each routing node in a n-ary ($n > 2$) tree formation. The TD-FALCON n-ary tree structure has the same feature of topological symmetry as TD-FALCON BTF, but may increase the size of the state and action spaces significantly. In addition, it may lack the flexibility of the binary tree formation in handling a varying type of the network topologies.

In this chapter, we use the network routing (NR) problem, a typical TMAS domain, as the research model and the benchmark problem for the comparative experiments. NR has been an important and well studied problem [89]. Various forms of network routing problems exist, including aircraft routing [92], transportation network routing [93] and "shoreline" vehicle routing and scheduling [94]. Our study in this work is based on a generic form of network routing problem, as defined in Section 5.2.

5.1 Related Work

Topology-based Multi-Agent Systems (TMAS) construct a special category of multi-agent domains, wherein agents interact with one another according to their spatial relationship. So far there are mainly four groups of cooperative strategies developed for TMAS

domains, namely operations research (OR), heuristic methods, Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs), genetic algorithms (GA), as well as reinforcement learning techniques. Subsequently we describe those existing methods in details.

In TMAS domains, operations research (OR) methods, particularly linear programming, have been used extensively. For example, Vannelli [95] presents a linear programming model to solve the global routing problem. Baldacci et al [96] present a lower bound derived from the linear programming (LP) relaxation of the new formulation to address the capacitated vehicle routing problem (CVRP). Chu et al [97] propose a new linear programming model and preliminary lower bounds based on graph transformation. Generally, the linear programming methods have shown the best possible use of available productive resources and improvement in the quality of decision making. Nevertheless, the LP methods may produce non-optimal solutions and the graphical method of LP is restricted to the condition of two variables [98].

Heuristic methods have been found effective in some TMAS cases. Grigor'eva [99] provides a class of heuristic algorithms for solving the routing problem. Zhu et al [100] explore various heuristic methods to solve the vehicle routing problem with time windows. Sun et al [101] present a heuristic algorithm appropriate for vehicle routing problem with multiple demand nodes in embedded equipment. Although the heuristic methods may be effective in some ad-hoc cases as mentioned above, they are still restricted to the environments of finite state space representations, fully observable environments, deterministic state transitions, as well as static environments.

More sophisticated optimization techniques based on Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs) can be extended to uncertain TMAS environments. Rathnasabapathy and Gmytrasiewicz [102] use partially observable Markov decision processes (POMDP's) as a basic framework to formalize network routing as a multi-agent decision problem. Han [103] proposes the localized adaptive QoS routing scheme using POMDP. However, those optimization techniques cannot scale well to complex supply chain environments [104, 105].

To overcome the shortcoming of the POMDP approaches, some genetic algorithms (GA) have been used in the network routing problem. Munetomo et al [106] propose an adaptive routing algorithm using genetic operators to realize an intelligent routing which directly observes communication latency of the routes. Al-Ghazal et al [107] propose an algorithm for improving routing in clustering algorithm based on both clusterhead gateway switching protocol (CGSR) and the mechanisms of a genetic algorithm (GA). However, those generic algorithms can not overcome the intrinsic drawback of slow convergence [108].

Reinforcement learning techniques based on the single agent strategy, wherein one autonomous agent is employed for each routing node in a TMAS domain, have also been proposed. Littman and Boyan [109] provide a self-adjusting algorithm for packet routing, in which a reinforcement learning module is embedded into each node of a switching network. Hiroyuki et al [110] propose a routing algorithm using ants computing and reinforcement learning method, which has been proven to give optimum resolution in stationary network traffic. Khodayari and Yazdanpanah [111] propose a reinforcement learning (RL) algorithm for packet routing in computer networks with an emphasis on different traffic conditions. They have shown that routing with an RL approach can result in shorter sending time and less congestion in traffic. Tongeron et al [112] find that the reinforcement learning agents can learn better policies than humans, but they do not always converge to the optimal policy. Despite the achievements of reinforcement learning techniques in TMAS domains, the stability of traditional reinforcement learning methods can be very poor [33]. Additionally, the state-of-the-art reinforcement learning methods have shown to be inflexible under varying scales of working environments [113, 114].

5.2 The Network Routing Problem

As shown in Figure 5.1, the task of NR is to send a number of packets from the *supply node* S to the *demand node* D, through a network composed of n ($n > 0$) one-way connected *transshipment nodes* $T = \{T_1, T_2, \ldots, T_n\}$, within a given period. Mathematically, a NR system can be formalized as a weakly connected, oriented, acyclic and simple graph $G =$

(V, A), where V denotes the set of all nodes and A denotes the set of all arcs. The nodes in the graph G include the supply node S, the demand node D and all the transshipment nodes in the NR system, so there is $V = \{S\} \cup \{D\} \cup T$. An arc refers to a one-way transfer path between two nodes. If there is an arc from node A to node B, then B is named as a *successor* of A. We define a node R as a *routing node* if $deg^+(R) > 1$. A routing node needs to route and send a packet among a few alternative transfer paths.

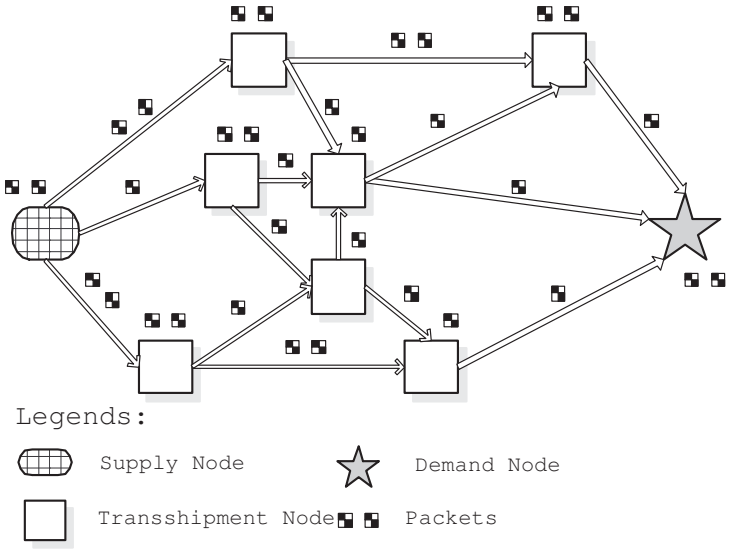


Figure 5.1: A network routing domain.

Capacities and transfer speeds among transshipment nodes can be different. Each transshipment node is equipped with a queue of specific size (capacity) to store the received packets. A traffic jam may occur if too many packets are queued beyond the capacity of a transshipment node. We define *hop* as a time circle within which a packet is transferred between two nodes, and multiple transshipment nodes can transfer a varying number of packets concurrently within a hop (transfer speed). A NR task is deemed as a *success* if all packets can be transferred to the demand node within the specified number of hops. If any traffic jam occurs or time is run out, the NR task is deemed as a $failure$.

5.3 Single Agent Strategy

A straightforward solution to the network routing problem is the single agent strategy, which deploys one agent for each routing node. This approach can be easily implemented and is commonly adopted [20, 115, 116]. However, as discussed earlier, since each agent has a different state space, scalability may become an issue. For the purpose of completeness and comparison, we describe the single agent strategy based on the TD-FALCON algorithm in this section.

5.3.1 Sensory Representation and Reward Scheme

In a NR environment, a TD-FALCON agent installed at a routing node receives sensory inputs from all of its successor transshipment nodes. The sensory inputs include the transfer speed, the queue volume, and the number of queued packets of each successor, as described below.

Transfer speed is defined as the number of packets transferred at each hop. For the purpose of input normalization as described in Section 3.3.4.1, we set a limit on the maximum transfer speed to be 1 packet per hop. Mathematically, the sensory input for the transfer speed is denoted by $S_t = \frac{1}{T}$, where T is the number of hops required to transfer a packet.

Queue volume is defined as the number of packets that a transshipment node can queue. For input normalization, we use the proportion from the queue volume C to the maximum queue volume C_{max} as a state input. For example, if a transshipment node has a queue volume of 8 and the maximum queue volume across all transshipment nodes is 10, the relevant sensory input should be 0.8. We denote the sensory input for the queue volume as $S_c = \frac{C}{C_{max}}$.

The number of queued packets counts the number of packets that are being queued at the successor. For normalization, we use the reciprocal of the number of the queued packets as a state input. For example, if there are five packets queued in a transshipment node, the relevant sensory input is 0.2. Mathematically, the sensory input is denoted by $S_q = \frac{1}{Q}$, where Q is the number of queued packets.

In summary, the sensory inputs that the routing node receives from a successor can be denoted by

$$S = (S_t, S_c, S_q). \tag{5.1}$$

Assume that a routing node R has m successors, known as TS_0, TS_1, ..., TS_{m-1}, then the action space of the routing node can be denoted by

$$A = \{a_0, a_1, \ldots, a_{m-1}\}, \tag{5.2}$$

where $a_I = 1$ if a_I corresponds to the action of transferring a packet to TS_I and $a_i = 0$ for $i \neq I$.

In the discretized form, the size of state space received from a successor is $N = |T| \cdot |C| \cdot |Q|$. If a routing node R has m successors, the overall size of the state space of R is

$$N^m \qquad where\ N = |T| \cdot |C| \cdot |Q|. \tag{5.3}$$

If m is a large number, a scalability problem emerges.

Moreover, as mentioned earlier, a major problem of the single TD-FALCON approach to the NR problem is the different state spaces required across the various routing nodes. Assume that a routing node R_M has m successors and another routing node R_N has n successors. On the condition of $m \neq n$, the TD-FALCON agents installed in the routing nodes R_M and R_N typically have a different architecture and thus knowledge sharing between the two agents is impossible. As a result, the overall number of cognitive nodes may increase significantly.

5.4 TD-FALCON Binary Tree Formation (BTF) Strategy

Based on the TD-FALCON model, we propose the use of TD-FALCON teams with the Binary Tree Formation (TD-FALCON BTF) strategy. Instead of using a single TD-FALCON agent, we equip a routing node with a number of TD-FALCON agents, which are assembled in a binary tree formation. For example, if the routing node R is to route and

send packets to its eight successor transshipment nodes (numbered from TS_0 to TS_7), we can build a three-level TD-FALCON BTF to receive and process sensory inputs, as shown in Figure 5.2.

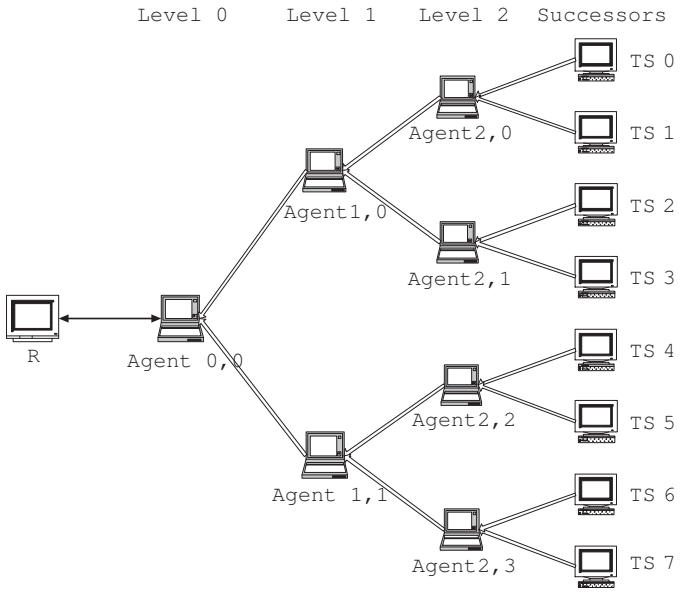


Figure 5.2: A TD-FALCON binary tree formation with eight successor transshipment nodes.

5.4.1 Working Mechanism of TD-FALCON BTF

The working mechanism of a n-level ($n \geq 1$) TD-FALCON BTF is described in Table 5.1 and 5.2. For the ease of description, we include the successor transshipment nodes as the leaf nodes of the binary tree. In the binary tree formation, the root node is in *Level 0*, the next level consisting of its two child nodes is *Level 1*, and finally the level of the leaf nodes (or the successor transshipment nodes) is *Level n-1*. We denote a TD-FALCON agent as $Agent_{p,q}$, where p ($0 \leq p \leq n-2$) is the level of the TD-FALCON agent in the binary tree, and q ($0 \leq q < 2^p$) is the ordinal of the TD-FALCON agent in Level p.

We give an example based on Figure 5.2 to demonstrate the TD-FALCON BTF working mechanism as follows.

1. $Agent_{p,q}\,(0 < p < n-1, 0 \leq q < 2^p)$ receives state inputs $S_{p,q} = (S_{p+1,2q}, S_{p+1,2q+1})$ from its two child nodes $Agent_{p+1,2q}$ and $Agent_{p+1,2q+1}$.
2. Based on the state vector $S_{p,q}$ and the action vector $A_{p,q} = \{a_{p+1,2q}, a_{p+1,2q+1}\}$, where $a_{p+1,i}$ $(2q \leq i \leq 2q+1)$ is the action of sending a packet to the successor recommended from $Agent_{p+1,i}$, $Agent_{p,q}$ selects an action $a_{p+1,j}\,(2q \leq j \leq 2q+1)$.
3. $Agent_{p,q}$ forwards the state inputs $S_{p+1,j}$ from $Agent_{p+1,j}$, to its parent node $Agent_{p-1,q/2}$.
4. Upon receiving a reward r from its parent $Agent_{p-1,q/2}$, - $Agent_{p,q}$ performs the learning process by using the state vector $S_{p,q}$, the action $a_{p+1,j}$, and the reward r. - $Agent_{p,q}$ transfers the reward r to $Agent_{p+1,j}$.

Table 5.1: The operational cycle of a branch-node agent $Agent_{p,q}$ in TD-FALCON BTF.

1. $Agent_{0,0}$ receives state inputs $S_{0,0} = (S_{1,0}, S_{1,1})$ from its two child nodes $Agent_{1,0}$ and $Agent_{1,1}$.
2. Based on the state inputs received and the action space $A_{0,0} = \{a_{1,0}, a_{1,1}\}$, where $a_{1,i}$ $(0 \leq i \leq 1)$ is the action of sending a packet to the successor recommended from $Agent_{1,i}$, $Agent_{0,0}$ selects an action $a_{1,j}\,(0 \leq j \leq 1)$.
3. After sending the packet to $TS_k\,(0 \leq k < 2^{n-1})$, which is the recommended successor from $Agent_{1,j}$, $Agent_{0,0}$ receives a reward r from the environment.
4. $Agent_{0,0}$ transfers the reward r to $Agent_{1,j}$.
5. $Agent_{0,0}$ performs learning by using the state vector $S_{0,0}$, the action $a_{1,j}$, and the reward r.

Table 5.2: The operational cycle of the root-node agent $Agent_{0,0}$ in TD-FALCON BTF.

1. $Agent_{2,0}$, $Agent_{2,1}$, $Agent_{2,2}$ and $Agent_{2,3}$ select the successor TS_1, TS_3, TS_5 and TS_6 respectively.

2. Subsequently, in Level 1, $Agent_{1,0}$ and $Agent_{1,1}$ take the action $a_{2,1}$ and $a_{2,2}$ respectively, which means that TS_3 and TS_5 are recommended to $Agent_{0,0}$.

3. $Agent_{0,0}$ takes the action $a_{1,1}$, sends a packet to the successor TS_5 and receives a reward r.

4. Based on the state inputs from TS_3 and TS_5, the action $a_{1,1}$, as well as the reward r, $Agent_{0,0}$ performs the learning process, and transfers the reward r to $Agent_{1,1}$.

5. Based on the state inputs from TS_5 and TS_6, the action $a_{2,2}$, as well as the reward r, $Agent_{1,1}$ performs the learning process, and transfers the reward r to $Agent_{2,2}$.

6. Based on the state inputs from TS_4 and TS_5, the action $a_{3,5}$, as well as the reward r, $Agent_{2,2}$ performs the learning process.

Since all TD-FALCON agents (including those from different routing nodes) have the same structures in the state and action spaces, their knowledge can be shared in the sense that a single set of cognitive nodes can be used and learned by the various agents concurrently.

5.4.2 Reward Scheme of TD-FALCON BTF

We adopt a hybrid reward scheme [3] in the TD-FALCON BTF algorithm. For the local rewards, we use the *Expected Transfer Time (ETT)* of a packet as a measure, which is defined as the number of hops that a packet is expected to be transferred from a transshipment node. Suppose that a packet has been transferred from a routing node R to a successor TS_j, which has a transfer speed of $\frac{1}{T}$. TS_j has the queue volume C and now there are Q packets in queue. Then there are two possible situations: (1) if $Q < C$, TS_j is not jammed, the ETT of TS_j is given by $ETT_j = T(Q+1)$; (2) if $Q \geq C$, the TS_j node is

jammed and the task is considered as a failure. Consequently, the ETT of TS_j should be assigned with a huge number, e.g. 999.

As for the global rewards, we use the *Overall Queued Packets (OQP)* to indicate the total number of packets queued in all transshipment nodes. Assume that there are altogether n transshipment nodes, where the mth transshipment node is marked as TS_m $(0 \leq m \leq n-1)$ and the number of packets queued in TS_m is marked as Q_m. The OQP of the entire network routing system can thus be computed as $OQP = \sum_{m=0}^{n-1} Q_m$.

Combining the local and global rewards, we are able to get a hybrid reward below for the routing node R after it sends a packet to the successor TS_j:

$$R = \frac{1}{ETT_j(1 + OQP)}. \tag{5.4}$$

Using the hybrid reward, a routing node can make a balance between the requirement of sending packets in a short time and minimizing the traffic jam in the entire NR domain, by distributing packets evenly among the successors.

5.4.3 Complexity Analysis of TD-FALCON BTF

If R is a routing node with m successors, the number of TD-FALCON agents required to construct a binary tree for R is

$$N = m - 1. \tag{5.5}$$

It is noticed that all TD-FALCON agents in a binary tree formation have a uniform state space (N^2), action space (2 choices) and reward scheme. They are thus fully symmetrical in terms of both architecture and learning algorithms. Adopting the knowledge sharing method proposed by Tan [18], all those TD-FALCON agents in the binary tree are able to share the same set of cognitive nodes. Hence only one set of cognitive nodes needs to be learned. Furthermore, the same set of cognitive nodes can also be shared by TD-FALCON agents from binary trees installed in other routing nodes, because all of them have the uniform TD-FALCON architecture.

Although there is a certain delay in making a routing decision when sensory inputs are

transmitted across different levels in a binary tree, the overall delay time is limited to $O(log_2^m)$, as there are only $\log_2 m$ levels in a m-node binary tree. The routing selection process can be conducted concurrently from multiple TD-FALCON agents in the same level of a binary tree, and so there is no delay within the level. Since the learning process can be performed concurrently among TD-FALCON agents from various levels of a binary tree, the time cost of the learning process in a TD-FALCON BTF is in the same order as the single agent strategy.

5.4.4 Comparison with TD-FALCON n-ary Tree Formation

In the most general case, it is possible to assemble a number of TD-FALCON agents as a n-ary tree (where $n > 2$) for a routing node. Having a topological symmetry, a n-ary tree can also implement the knowledge sharing mechanism among all TD-FALCON agents. Subsequently we analyze the performance of the TD-FALCON n-ary Tree Formation in regard to the number of agents required, the delay time, the topological flexibility, as well as the sizes of the state and action spaces.

Lemma: A n-ary tree structure ($n \leq m$) installed in a routing node with m successors is composed of $\frac{m-1}{n-1}$ TD-FALCON agents, fewer than the number of agents needed in a binary tree formation.

Proof: Assume that a routing node R has m successors. It can be known that there are $\log_n m$ levels of nodes in the tree in total. We assign $P = \log_n m$, and then the total number of nodes in the n-ary tree is

$$N = 1 + n + n^2 + \ldots + n^{P-1} = \frac{n^P - 1}{n - 1} = \frac{m - 1}{n - 1}. \tag{5.6}$$

If we take $n = 2$, the total number of nodes is $m - 1$. This reduces to Equation 5.5.

[end of proof].

Equation 5.6 highlights a trend that with the growth of n, the number of TD-FALCON nodes in a n-ary tree is decreased in an inversely proportional manner, meaning that a n-ary tree TD-FALCON structure has the cost of $O(\frac{1}{n})$ in the number of nodes.

In addition to having fewer TD-FALCON agents in construction, a n-ary tree structure shows another advantage of having the delay time (log_n^m) in a routing node's performing process, less than a binary tree formation (log_2^m). Nevertheless, such an advantage can be minimized if a powerful computing tool is used.

A n-ary tree structure however has two big problems. Firstly, if the number (m) of successors of a routing node is not in the order of 2^n, it is difficult to build a symmetrical n-ary tree structure. For example, if $n = 4$ and $m = 9$, establishing a standard $4 - ary$ tree with topological symmetry is a tough task. In that case, we have to maintain the symmetry of the tree structure by adding some special "dummy" TD-FALCON agents, which have all the inputs and outputs as 0. However, using the fabricated values generated from those "dummy" TD-FALCON agents will definitely deteriorate the accuracy of such a system. On the contrary, a binary tree can keep the topological symmetry in a routing node with any number of successors. Subquently we provide the proof for this point, using the induction method.

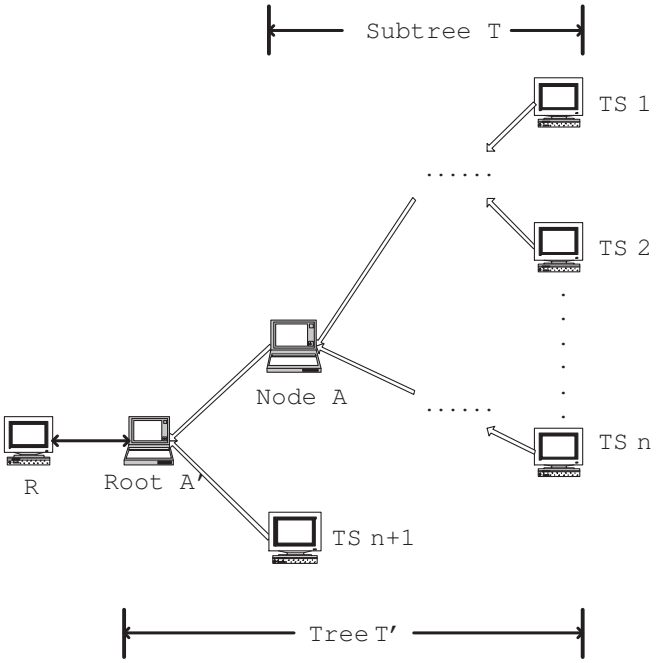


Figure 5.3: Construction of a binary tree with $n + 1$ leaf nodes.

Lemma: A binary tree can always be constructed in a routing node with any number n ($n \geq 2$) of successors.

Proof: The proof of the lemma consists of three parts as follows.

Part I: It is evident that a binary tree can be built when $n = 2$.

Part II: Assume that a binary tree can be constructed in a routing node with n ($n \geq 2$) successors.

Part III: If a routing node R has $n+1$ successors, known as TS_0, TS_1, ..., TS_n and TS_{n+1}, according to *Part II*, a binary tree T can be constructed with the n successors TS_0, TS_1, ..., TS_n as the leaf nodes, and the root of the tree T is marked as A. Create a new node A'. By establishing an edge from A' to A and an edge from A' to TS_{n+1}, we are able to build a new binary tree T', wherein A' is the root, A and TS_{n+1} are respectively the left child and right child of the root A', and T is the left subtree of T'. Thus the binary tree T' is exactly constructed for the routing node R and the proof is accomplished. The construction process of the new binary tree T' is also illustrated in Figure 5.3.

[end of proof].

The second problem of a n-ary tree is the significant increment in state and action spaces. The number of actions that a TD-FALCON agent can select is increased from 2 in a binary tree to n in a n-ary tree, so the increment in the action space is linear. According to Equation 5.3, the state space is exponentially increased from N^2 in a binary tree to N^n in a n-ary tree.

Consider a routing node with m successors, the time and space complexities of TD-FALCON teams in single agent strategy, binary tree formation, and n-ary tree formation are summarized in Table 5.3, wherein N represents the dimension of the state vector received from one successor and m is the number of successors for a routing node.

parameter	binary tree	n-ary tree	single agent
Number of cognitive nodes	$O(N^2)$	$O(N^n)$	$O(N^m)$
Delay time	$O(log_2^m)$	$O(log_n^m)$	$O(1)$
Size of state space	$O(N^2)$	$O(N^n)$	$O(N^m)$
Size of action space	$O(1)$	$O(n)$	$O(m)$

Table 5.3: Comparing the time and space complexities of TD-FALCON teams in single agent strategy, binary tree formation, and n-ary tree formation.

5.5 Experimental Results

In this section, we perform a series of experiments in a generic network routing task to compare the performance of TD-FALCON teams in various configurations as well as a classical linear programming (LP) method. We also report experiments wherein we test the scalability of the TD-FALCON BTF algorithm in various network and traffic conditions.

In all experiments, TD-FALCON agents comply with the bounded Q-learning rule (Equation 3.14) and use the following parameter values: the learning rate α is fixed at 0.5, the discount factor γ is set to 0.95, and the initial Q-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 0.0004.

5.5.1 Comparing TD-FALCON Teams in Various Formations

In this group of experiments, we test the performance of TD-FALCON teams in the single agent, binary tree, trinary tree, and 5-ary tree formations, under 50 packets and 20 transshipment nodes. The transfer speeds can be 0.2, 0.333 or 0.5 packets per hop, and are distributed randomly across the transshipment nodes. We use various configurations of the 20 transshipment nodes to average the performance. A few samples of the configurations are shown in Figure 5.4. The purpose of the experiments is to compare the performance of various TD-FALCON based strategies.

Figure 5.5 shows the success rates of TD-FALCON teams in the single agent, binary

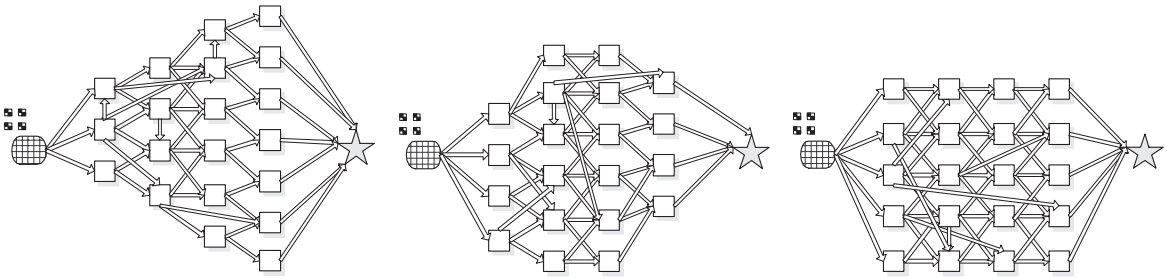


Figure 5.4: Network layout samples of experiments.

tree, trinary tree, and 5-ary tree formations, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It can be seen that all the teams can eventually achieve 100% success rate. However, among all of them, the team in the binary tree formation has the highest convergence rates. The BTF team can achieve more than 90% success rates after 1,300 trials. On the other hand, the single TD-FALCON team has the lowest convergence rate, not achieving more than 90% success rate until about 1,800 trials.

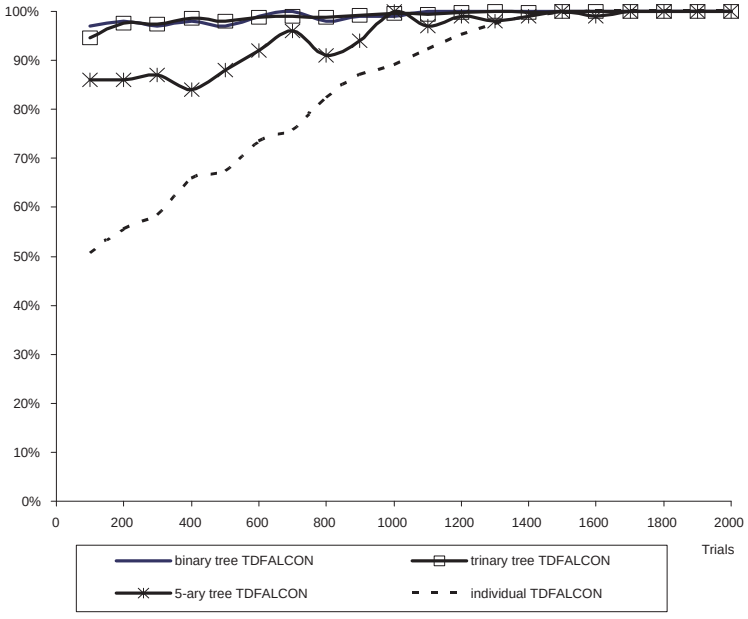


Figure 5.5: Success rates of various TD-FALCON based strategies.

Figure 5.6 shows the numbers of hops incurred by TD-FALCON teams in the single agent, binary tree, trinary tree, and 5-ary tree formations, averaged at 100-trial intervals

across 2,000 trials on the NR task. The single TD-FALCON team notably needs more hops than those tree structure teams to transfer the 50 packets. The TD-FALCON BTF team and the trinary tree TD-FALCON team have the least number of hops. After 2,000 trials, both of the teams averagely takes about 53 hops to fulfill a task, in contrast to 54.7 hops from the single TD-FALCON team.

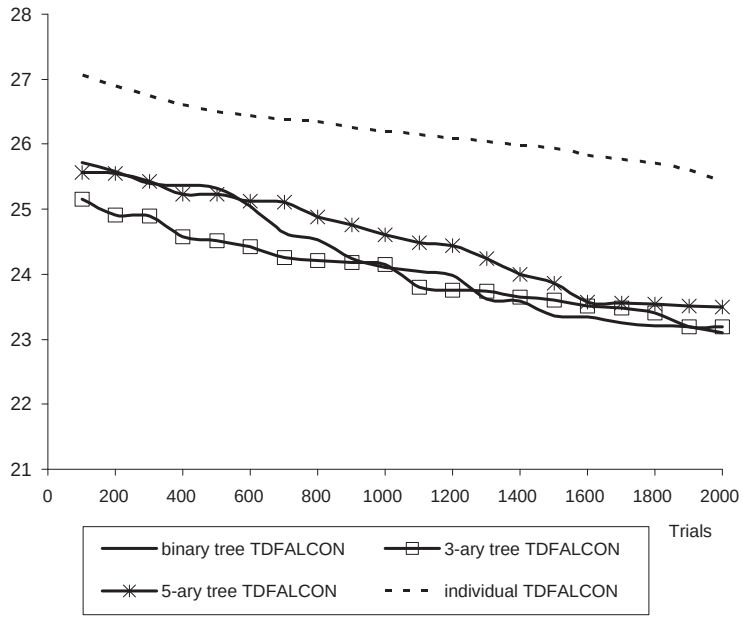


Figure 5.6: Numbers of hops of various TD-FALCON based strategies.

Figure 5.7 shows the numbers of cognitive nodes created by TD-FALCON teams in the single agent, binary tree, trinary tree, and 5-ary tree formations, averaged at 100-trial intervals across 2,000 trials on the NR task. We can notice that the number of cognitive nodes increases with the number n of TD-FALCON n-ary trees. For example, the TD-FALCON 5-ary tree team on the average produces about 476 cognitive nodes after 2,000 trials, nearly 10 times (one order of magnitude) more than 48.3 cognitive nodes generated from the TD-FALCON BTF team. It can also be noticed that the single TD-FALCON team generates much more cognitive nodes than the TD-FALCON tree teams. After 2,000 trials, the single TD-FALCON team produces about 6,611 cognitive nodes on average, over 130 times (two orders of magnitude) more than the TD-FALCON BTF team.

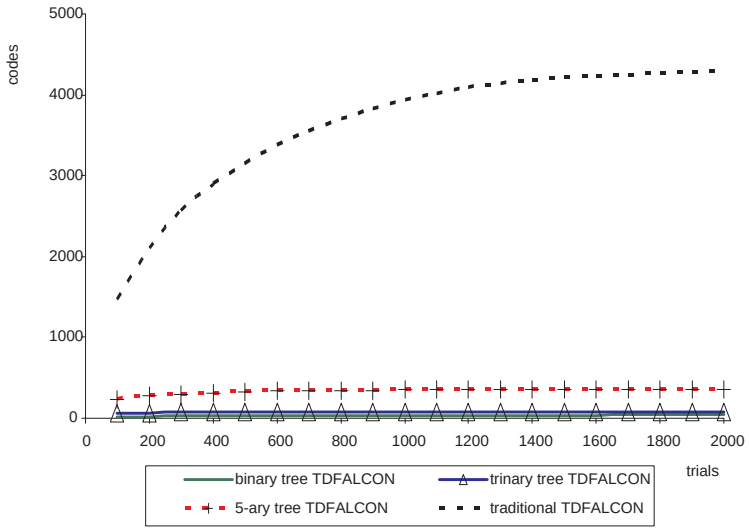


Figure 5.7: Numbers of cognitive nodes of various TD-FALCON based strategies.

Figure 5.8 demonstrates the running times (seconds) per trial of TD-FALCON teams in the single agent, binary tree, trinary tree, and 5-ary tree formations, across 2,000 trials, in the NR system. We can notice an obvious trend from all tree structure teams that the running time is increased with the growth of cognitive nodes. The other noticeable trend is that a TD-FALCON n-ary tree system runs slower with the growth of the number n. For example, after 2,000 trials, the TD-FALCON 5-ary tree team spends about 7.59 seconds in running one trial, almost 12 times (one order of magnitude) slower than the TD-FALCON BTF team, which uses only 0.612 seconds for one trial. It can be noticed that the single TD-FALCON team spends tremendously more time than the TD-FALCON teams with tree formations in running the system. After 2,000 trials, the single TD-FALCON team needs to take more than 98 seconds to run one trial, about 160 times (two orders of magnitude) slower than the TD-FALCON BTF team.

5.5.2 Comparative Experiments with Linear Programming

In this group of the experiments, we compare the performance between the TD-FALCON BTF team and a linear programming (LP) method, under the same configuration of 50

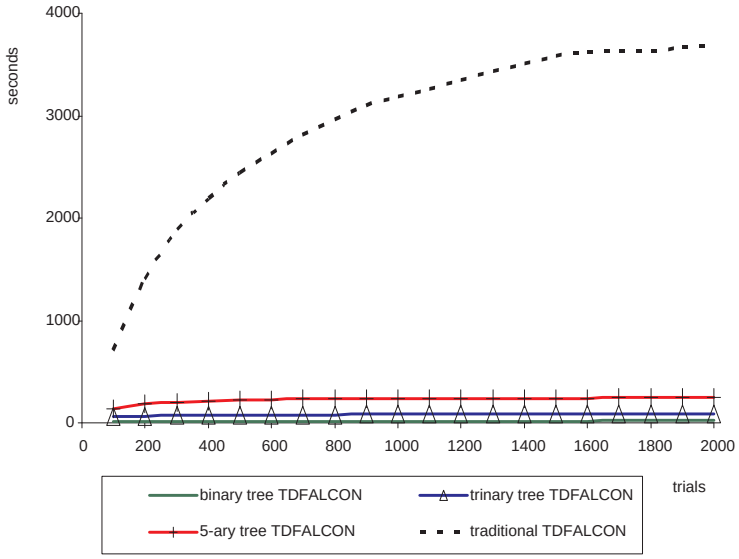


Figure 5.8: Running times (seconds) of various TD-FALCON based strategies.

packets and 20 transshipment nodes.

In the LP method, we install each routing node in the NR domain with a revised simplex algorithm [117, 118], in which the matrix form in the formation of Equation 5.7 and 5.8 is used at each iteration of the simplex algorithm.

$$\text{minimize} \qquad \mathbf{c}^T\mathbf{x} \tag{5.7}$$

$$\text{subject to} \qquad \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq 0 \tag{5.8}$$

In perspective of a routing node, we define the network routing problem in the formation of linear programming as follows. Assume that a routing node R needs to dispatch $n\,(n > 0)$ packets to $m\,(m > 0)$ successors, marked as $TS_1, TS_2, \cdots, TS_m$, and the successor $TS_p\,(1 \leq p \leq m)$ has the queue volume C_p, the transfer speed T_p and the number of queued packets Q_p. Assume that the routing node R will dispatch $x_1, x_2, \cdots, x_n$ packets to the successor $TS_1, TS_2, \cdots, TS_m$ respectively, and there is $\sum_{p=1}^{m} x_p = n$. We define *Expected Clearance*

Time (ECT) of a successor $TS_p\,(1 \leq p \leq m)$ as

$$ECT_p = \frac{x_p + Q_p}{T_p} = \frac{x_p}{T_p} + \frac{Q_p}{T_p}. \tag{5.9}$$

The *Average ECT (AECT)* of the m successors of the routing node R is defined as

$$AECT = \frac{\sum_{p=1}^{m} ECT_p}{m} = \sum_{p=1}^{m} \frac{x_p}{T_p m} + \sum_{p=1}^{m} \frac{Q_p}{T_p m}. \tag{5.10}$$

In the *standard form* of a linear programming problem, the network routing problem of the routing node R can be described as

$$\text{minimize} \qquad \sum_{p=1}^{m} \frac{x_p}{T_p m} + \sum_{p=1}^{m} \frac{Q_p}{T_p m} \tag{5.11}$$

$$\text{subject to} \qquad x_p + Q_p \leq C_p\,(1 \leq p \leq m) \tag{5.12}$$

$$x_p \geq 0\,(1 \leq p \leq m). \tag{5.13}$$

Taking

$$\mathbf{c} = \begin{bmatrix} \frac{1}{T_1 m} \\ \frac{1}{T_2 m} \\ \cdots \\ \frac{1}{T_m m} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}, and\, \mathbf{b} = \begin{bmatrix} C_1 - Q_1 \\ C_2 - Q_2 \\ \cdots \\ C_m - Q_m \end{bmatrix}$$

we can get the matrix form of the linear programming problem as Equation 5.7 and 5.8.

Note that it is very difficult to implement the global reward mechanism for the LP team, because each LP agent is installed locally on a routing node. As its functionality can only cover its successors, it does not have the capacity to cope with the global information within its own standard form format. If we employ a global LP agent to collect and process all data from the entire system, this configuration definitely becomes a single agent problem instead of a multi-agent system.

Figure 5.9 shows the success rates of the TD-FALCON BTF and the LP teams, averaged at 100-trial intervals across 2,000 trials, in the NR system. It is obvious that the TD-FALCON BTF team can learn much faster than the LP team. Initially, the LP team has around 70% success rate, while the TD-FALCON BTF team has only 12% success rate.

However, after 2,000 trials, the TD-FALCON BTF team can achieve 100% success rate, while the LP team keeps at the same level.

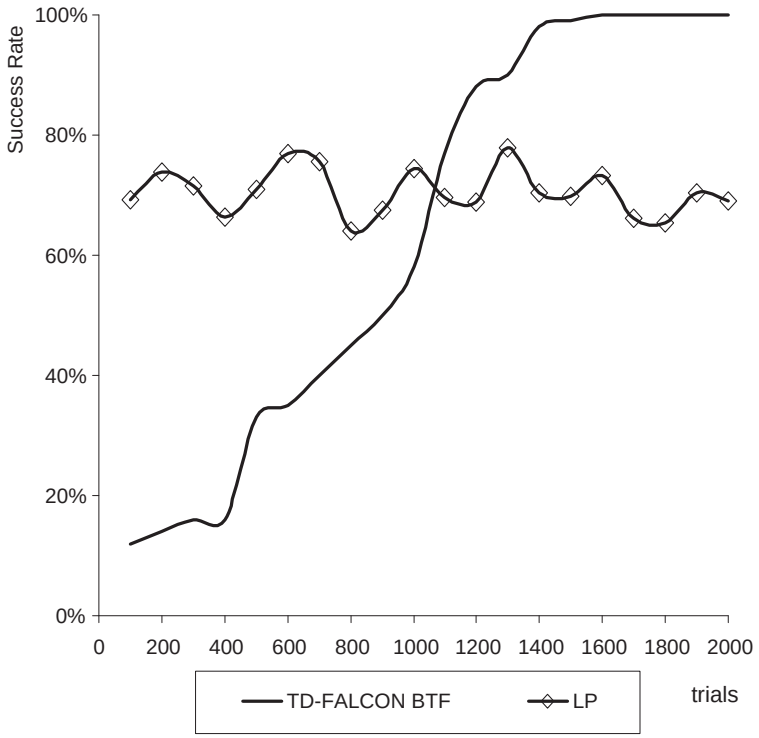


Figure 5.9: Success rates of TD-FALCON BTF and LP teams.

Figure 5.10 shows the numbers of hops of the TD-FALCON BTF and the LP teams, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It is witnessed that the number of hops required in the TD-FALCON BTF team decreases gradually over the trials, but the curve of the LP team is basically flat with some fluctuations. This means that the TD-FALCON BTF team is able to improve the efficiency over time. Compared with the LP team, the TD-FALCON BTF team takes fewer hops to fulfil the task. After 2,000 trials, the average number of hops of the TD-FALCON BTF team is 52.9, in contrast to 60.8 hops of the LP team.

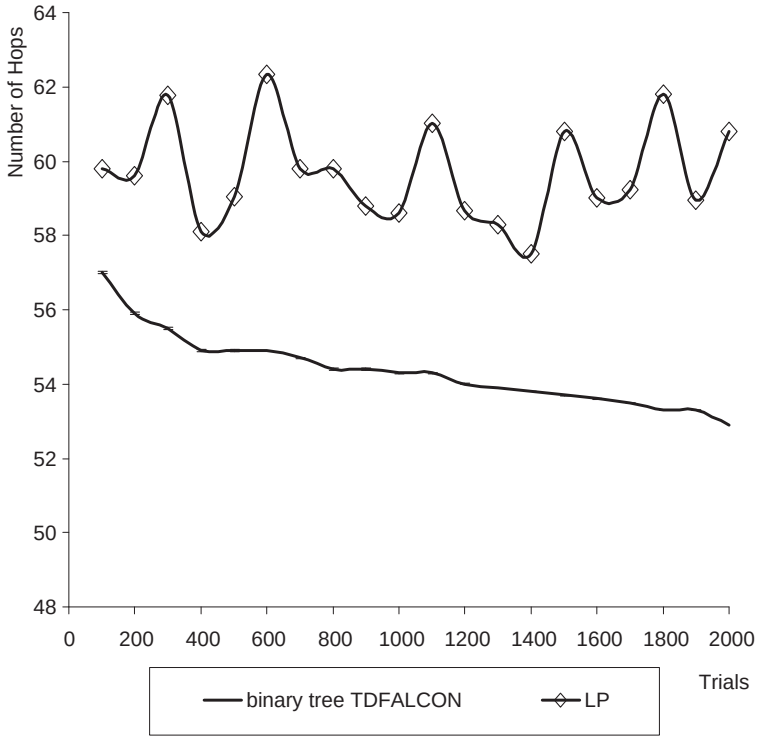


Figure 5.10: Numbers of hops of the TD-FALCON BTF and the LP teams.

5.5.3 Experimenting with Varying Number of Packets

In this group of experiments, we test the performance of the TD-FALCON BTF team in a network composed of 20 transshipment nodes, with 50, 80 and 100 packets to transfer respectively. The purpose of this group of experiments is to test the scalability of a TD-FALCON BTF team transferring a varying number of packets.

Figure 5.11 shows the success rates of a TD-FALCON BTF team transferring 50, 80 and 100 packets respectively, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It is witnessed that the TD-FALCON BTF team can eventually converge, regardless of the number of packets being transferred. However, it is noticed that the convergence becomes a bit slower with the increase in the number of packets. The TD-FALCON BTF team achieves more than 90% success rate within 1,300, 1,400, and 1,500 trials, for the tasks of transferring 50, 80, and 100 transferred packets respectively.

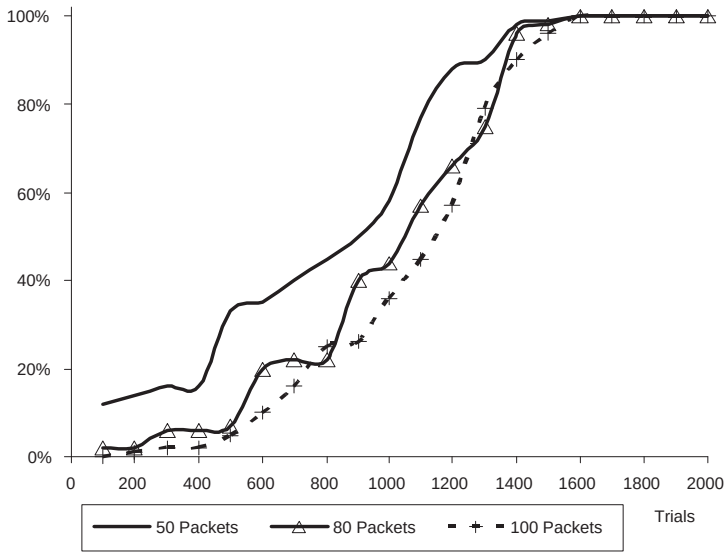


Figure 5.11: Success rates of a TD-FALCON BTF team transferring a varying number of packets.

Figure 5.12 shows the numbers of hops of a TD-FALCON BTF team transferring 50, 80 and 100 packets respectively, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It is obvious that the number of hops increases proportionally with the number of the packets to be transferred. After 2,000 trials, the average numbers of hops required to transfer 50, 80, and 100 packets are averagely 52.9, 82.5, and 102.6 respectively. It can also be noticed that the number of hops decreases through trials, though in a slow pace.

Figure 5.13 shows the numbers of cognitive nodes created from a TD-FALCON BTF team transferring 50, 80 and 100 packets respectively, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It is noticed that the number of cognitive nodes increases when more packets are transferred, but the increase is limited. After 2,000 trials, the numbers of cognitive nodes developed for transferring 50, 80 and 100 packets are on average 48.3, 54.3 and 57.5 respectively.

Figure 5.14 shows the running times (seconds) of a TD-FALCON BTF team transferring 50, 80 and 100 packets respectively, across 2,000 trials, in the NR domain. It is obvious

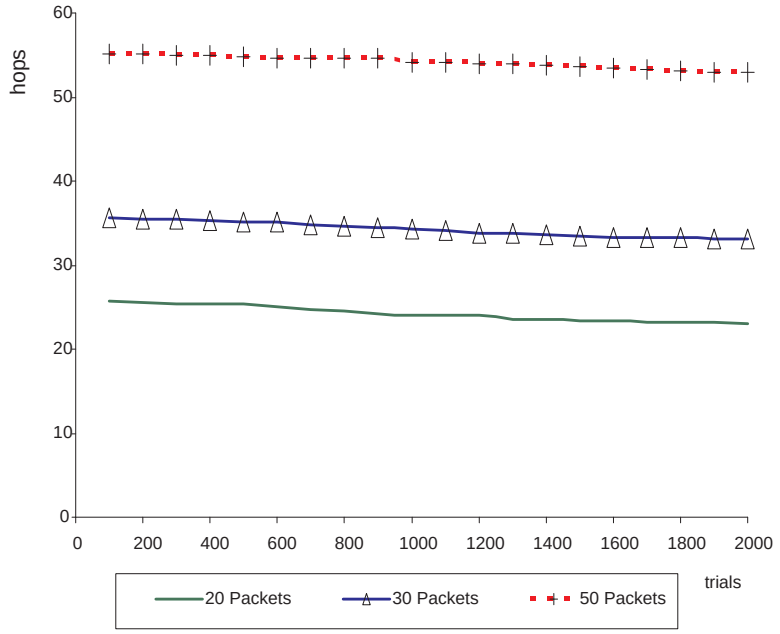


Figure 5.12: Numbers of hops of a TD-FALCON BTF team transferring a varying number of packets.

that the TD-FALCON BTF system has to run a longer time with the growth of the number of packets to be transferred. After 2,000 trials, the running times per trial of transferring 50, 80 and 100 packets are 0.612, 1.050 and 1.332 seconds respectively.

5.5.4 Experimenting with Varying Number of Transshipment Nodes

In this group of experiments, we test the performance of a TD-FALCON BTF team transferring 50 packets, in NR networks composed of 20, 30 and 50 transshipment nodes respectively. The purpose of this group of experiments is to test the scalability of the TD-FALCON BTF algorithm upon NR networks with a varying number of transshipment nodes.

Figure 5.15 shows the success rates of a TD-FALCON BTF team transferring 50 packets, in NR networks with 20, 30 and 50 transshipment nodes respectively, averaged at 100-trial intervals across 2,000 trials. It is noticed that the TD-FALCON BTF system can eventually

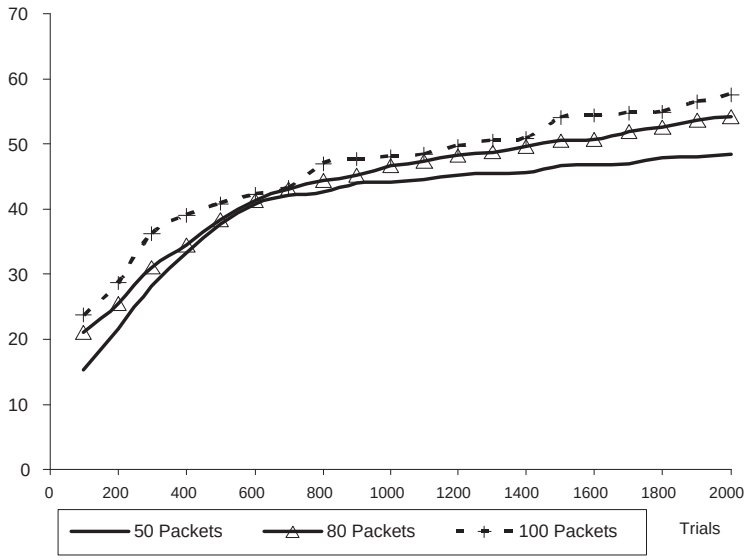


Figure 5.13: Numbers of cognitive nodes of a TD-FALCON BTF team transferring a varying number of packets.

achieve high success rates in NR networks with a varying number of transshipment nodes. After about 1,400 trials, the TD-FALCON BTF team can achieve more than 90% success rate in various network environments.

Figure 5.16 shows the numbers of hops of a TD-FALCON BTF team transferring 50 packets, in NR networks with 20, 30 and 50 transshipment nodes respectively, averaged at 100-trial intervals across 2,000 trials. It is evident that the number of hops is increased with the growth of the number of the transshipment nodes in the NR networks. After 2,000 trials, the numbers of hops in NR networks with 20, 30 and 50 transshipment nodes are about 52.9, 54.1 and 57.1 respectively. We can notice that the number of hops is declined gradually through trials in each NR network.

Figure 5.17 shows the numbers of cognitive nodes created from a TD-FALCON BTF team transferring 50 packets, in NR networks with 20, 30 and 50 transshipment nodes respectively, averaged at 100-trial intervals across 2,000 trials. It is noticed that cognitive nodes are increased with the growth of transshipment nodes, but the increment is limited.

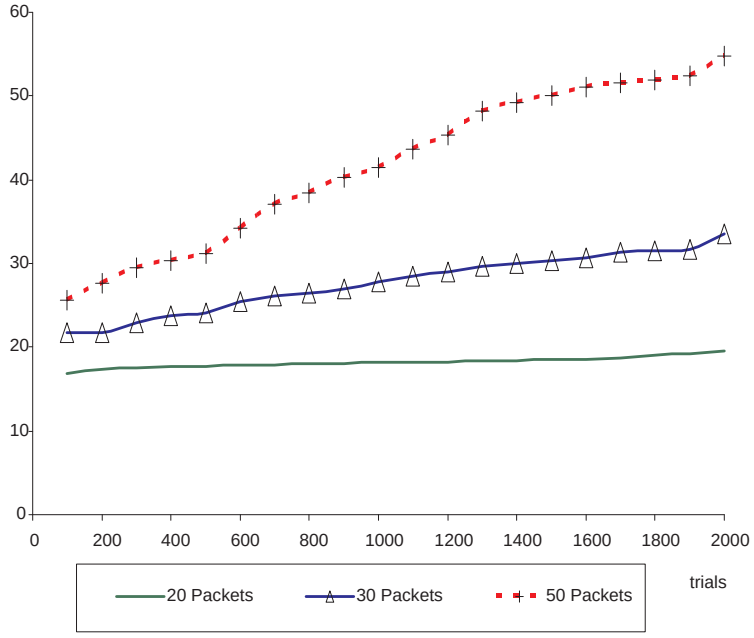


Figure 5.14: Running times (seconds) of a TD-FALCON BTF team transferring a varying number of packets.

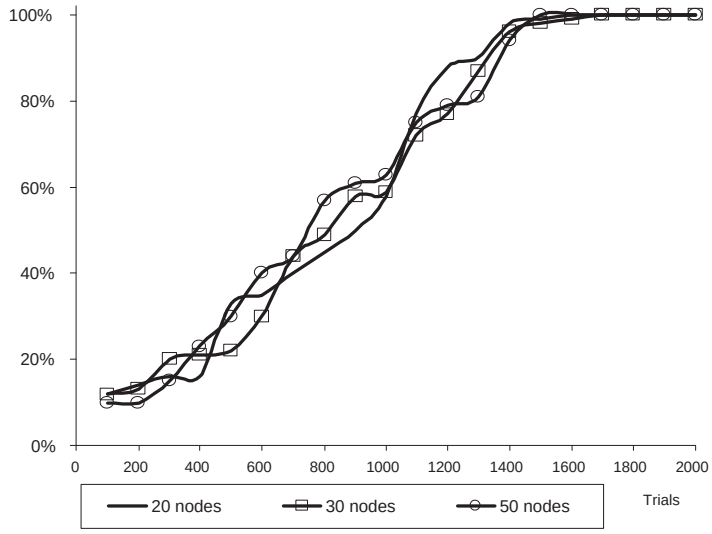


Figure 5.15: Success rates of a TD-FALCON BTF team in NR networks with a varying number of transshipment nodes.

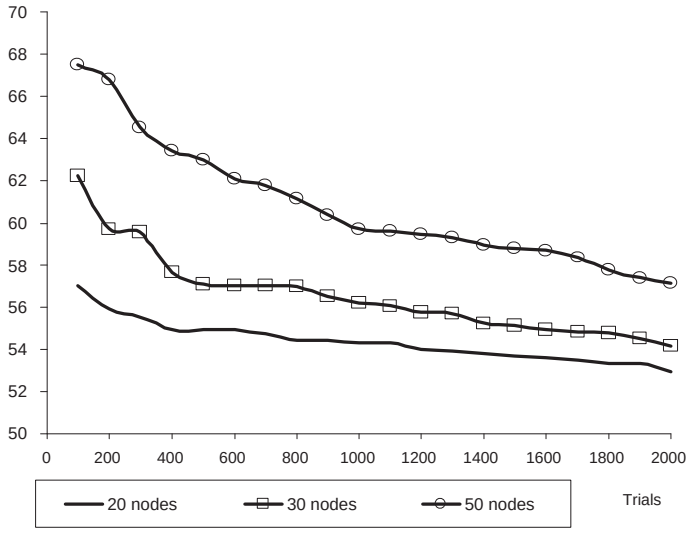


Figure 5.16: Numbers of hops of a TD-FALCON BTF team in NR networks with a varying number of transshipment nodes.

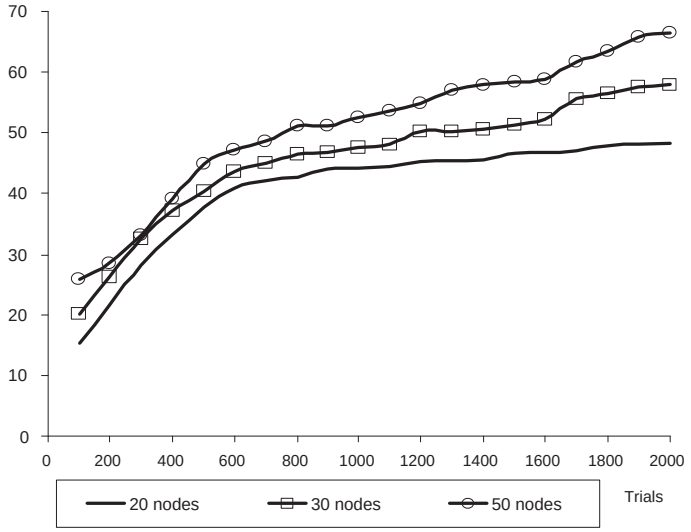


Figure 5.17: Numbers of cognitive nodes of a TD-FALCON BTF team in NR networks with a varying number of transshipment nodes.

After 2,000 trials, the numbers of cognitive nodes developed for NR networks with 20, 30 and 50 transshipment nodes are on average 48.3, 58.0 and 66.4 respectively.

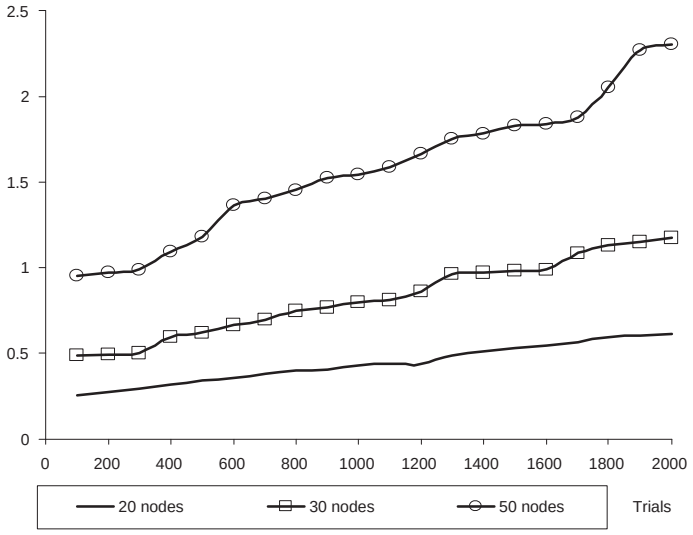


Figure 5.18: Running times (seconds) of a TD-FALCON BTF team in NR networks with a varying number of transshipment nodes.

Figure 5.18 shows the running times (seconds) of a TD-FALCON BTF team transferring 50 packets, in NR networks with 20, 30 and 50 transshipment nodes respectively, averaged at 100-trial intervals across 2,000 trials. It is noticed that a TD-FALCON BTF system spends less time in running within a NR network structure composed of fewer transshipment nodes. After 2,000 trials, the running times of the TD-FALCON BTF team under 20, 30 and 50 transshipment nodes are respectively 0.612, 1.177 and 2.306 seconds per trial.

5.6 Summary

In this chapter, we study a special group of multi-agent domains, namely topology-based multi-agent systems (TMAS), wherein each individual agent interacts only with its neighbors according to their spatial relationship. Network routing (NR) is a classical example of TMAS and used as a domain of study in this chapter.

To address the scalability issue in TMAS, we propose the use of TD-FALCON teams with the Binary Tree Formation (TD-FALCON BTF) strategy, wherein a routing node is

equipped with a number of TD-FALCON agents arranged in a binary tree formation. The TD-FALCON BTF strategy has two main points: (1) the binary tree formation can convert multiple state input sources to uniform binary input sources; (2) All TD-FALCON agents (even those under different routing nodes) are able to share the same set of cognitive nodes.

The complexity analysis of the TD-FALCON BTF. If a routing node has m successors, the number of TD-FALCON agents to construct the binary free formation should be $N = m - 1$. Each TD-FALCON agent in the binary tree has a uniform state space (N^2) and action space (2 choices). The overall delay time during the action selection process across the binary tree structure is limited to $O(log_2^m)$.

It is possible to organize a number of TD-FALCON agents as a n-ary tree (where $n > 2$), but the n-ary tree structure has two big issues: (1) it is very inflexible to construct a n-ary tree in front of any number of successors; (2) the n-ary tree structure can see a significant increment in state and action spaces.

Three groups of experiments have been organized to test the performance of the TD-FALCON BTF team. In the first group of experiments, the overall performance of TD-FALCON binary tree and trinary tree teams are at the high level, in contrast to poor performance of the sing agent and TD-FALCON 5-ary tree teams. The second group of experiments shows that the TD-FALCON BTF team remarkably outperforms the LP team. The results in the third group of experiments indicate that the TD-FALCON BTF team can still function and adapt well under the scaled-up network domain, but the convergence may be slower.

Chapter 6

Conclusion

6.1 Summary of Contributions

Through this dissertation work, we have made significant achievements in the development of multi-agent cooperative learning systems. The accomplished works include the development and deployment of a network architecture that is able to perform online and incremental learning in real-time multi-agent environment, the development of cooperative strategies that are able to compress the state space, and a special topological arrangement of agents for implementing the knowledge sharing mechanism to improve the convergence rate of agents. Those achievements are described below in details.

1. **Development of the TD-FALCON model.** We have developed a fusion architecture, known as Fusion Architecture for Learning, COgnition, and Navigation (FALCON), for learning multi-modal mappings across states, actions, and rewards [8]. Using a special category field to store cognitive nodes, the FALCON architecture is able to perform an online and incremental learning in a rapidly changing multi-agent environment. This is a great progress in multi-agent reinforcement learning, because the traditional reinforcement learning approaches, such as multi-agent Q-learning [11], the gradient descent based backpropagation (BP) learning algorithm [37] and so on, may become unstable in a multi-agent environment with large and/or continuous

state and action spaces. The FALCON architecture can also effectively control the state space by employing the template matching mechanism. This advantage enables a FALCON system to function well in a complex multi-agent task. A FALCON architecture can be extended into a TD-FALCON system, after being incorporated with the temporal difference (TD) algorithm. The TD-FALCON system is able to estimate and learn value functions without immediate reward signals, making an agent more workable in real-world multi-agent environments. The above-mentioned advantages of the TD-FALCON architecture have improved the adaptability of agents to such a high level that a TD-FALCON agent team without any cooperative strategies can even function well in a generic multi-agent system, wherein each agent is required to fulfil a task individually. This is demonstrated from experimental results of a typical benchmark problem, namely the multi-agent minefield navigation task. The experimental results also show that the TD-FALCON agents can significantly outperform the state-of-the-art backpropagation (BP) and the resilient propogation (RPROP) reinforcement learners, regardless of whether there are any immediate reward signals or not.

2. **Development of cooperative strategies for state space compression and scaling up.** We have worked out several cooperative strategies to significantly relieve the scalability issue in a complex multi-agent system by compressing the state space. This can help to relieve the long-term scalability issue [11] in multi-agent cooperative reinforcement learning. Failing to find out an effective way to summarize the state space of multiple agents, the previous research work can not be used in an environment with large and/or continuous action and state spaces. We have made efforts to change the situation by developing the center of agent team (CAT) cooperative strategy. The essence of the CAT strategy is that the state spaces from all agents can be summarized or compressed, as if they were generated from a "virtual" agent, which is defined as the centre of all agents [32, 31, 39]. We have conducted experiments using the CAT strategy in a typical predator/prey pursuit task, a multi-agent system which requires multiple agents to cooperate tightly to encircle a moving

target. The experimental results distinctly show the edge of the CAT team over the non-cooperating team. Based on the CAT cooperative strategy, we further develop a neighboring-agent mechanism (NAM), wherein the weak input signals are blocked, making the cooperative strategy more effective. The principle of the NAM strategy is that the summarized state space of an agent team can be further reduced by discarding those weak input signals. We have used the herding game, which is a multi-agent domain more complex than the predator/prey pursuit task, as a benchmark problem to test the performance of the NAM stragegy. In the herding game, a few agents (shepherds) need to not only encircle a moving target (sheep), but also force it to enter a specified corral within a grass land. The experimental results demonstrate that the NAM team obviously outperforms the CAT team, especially in term of the number of cognitive nodes, implicating the significant reduction in the overall state space.

3. **Propose a TD-FALCON Binary Tree Formation (BTF) for the TMAS domains.** We have used the topology of a group of agents to relieve the scalability issue in a special multi-agent domain, namely topology-based multi-agent system (TMAS), wherein agents interact with each other according to their spatial relationship. It is difficult to use the traditional single agent strategy in a TMAS environment, because each agent may have a state space with different size, due to the network complexity of the TMAS domain. In that case, each agent has to develop a different set of cognitive nodes, thus may delay the convergence process as knowledge sharing across multiple agents becomes impossible. To make things worse, an agent may have a large number of spatial relationships with other agents, causing a scalability issue in a TMAS domain with complex network topology. Our solution of arranging a group of TD-FALCON agents in a binary tree formation (BTF) exactly addresses the above-mentioned two issues. At first, the TD-FALCON BTF strategy can provide a uniform state space for all agents across an entire TMAS domain. Secondly, the TD-FALCON BTF strategy is able to convert a multi-source problem to a two-source problem, maintaining a constant and limited number of state input vectors. Com-

parative experiments in the network routing task, which is a typical TMAS domain, demonstrate that the proposed TD-FALCON BTF strategy can significantly outplay the other cooperative strategies, including the single agent strategy, the n-ary ($n > 2$) tree formations, and a classical linear programming algorithm.

6.2 Future Works

Although we have made some progress in the field of multi-agent cooperative reinforcement learning, there is still plenty of room for further investigations, especially on the aspects of the heterogeneous multi-agent domains, extensions to the n-level modeling agents and the systematization of TD-FALCON parameters and reward scheme. The major outstanding research issues and possible future work are detailed as follows.

1. **Extension to heterogeneous multi-agent domains.** For the time being our research works are restricted to homogeneous domains, wherein all agents are symmetrical, i.e., they have same functionalities and roles. However, there are wide ranges of heterogeneous multi-agent domains in real world. For example, a soccer game can be regarded as a heterogeneous domain, wherein the goal keeper and other players have different functionalities and roles. A network system is also a typical heterogeneous environment, as routers, bridges and switches in the network provide quite different functionalities: bridges and switches are just used for transferring data packets; routers can not only transfer data, but also sort the data frames and control the data flow. In a heterogeneous domain, the TD-FALCON BTF algorithm can still be used to reduce the state space of an agent, but how to apply the knowledge sharing mechanism across agents with different functionalities remains a great challenge.

2. **Extension to n-level modeling agents.** So far our work is focused on the 0-level modeling agent, as it is the most efficient one among all teammate modelings. In a 0-level modeling, an agent regards any other agent as nothing more than a part of the changing environment. However, the 0-level modeling can not cover all multi-agent

cases. For example, to have a good cooperation in a soccer game, a player needs to not only know the positions, the running speeds and the directions of his partners, but also understand their intentions in a short time. Therefore, the TD-FALCON based cooperative learning system must be enhanced to adapt to multi-agent environments with the n-level ($n > 0$) modeling. At present, we are able to encode the state space of other agents into the state inputs of the TD-FALCON performing and learning process of an agent. In the future work of a n-level modeling multi-agent environment, the intention, teammate modeling and even the cooperative strategy of other agents should also be encoded into the state inputs of a TD-FALCON learning agent. How to encode such complex information and meanwhile keep a high scalability in a multi-agent environment with n-level modeling is a tough task in our future work.

3. **Systematizing the selection of model parameters and reward scheme.** Presently, we are able to find out the appropriate parameters (e.g., choice parameter α, learning rate β, baseline vigilance parameter $\bar{\rho}$, discount parameter γ etc.) for the TD-FALCON learning process and the reward scheme in various multi-agent environments, but not until after many rounds of trials. Consequently, a lot of time will be wasted in optimizing those parameters. In addition, for the time being we have no way to verify whether the selected parameters and reward scheme are exactly the best choice, and it is impossible for us to try out all combinations of parameters to check. Moreover, even the originally optimal parameters and reward scheme may become void as long as there is a minor change in the environment settings, for example, the expansion of a pursuit game field from 16 by 16 to 17 by 17. It is also possible that the parameters must be tuned even across the trials of the TD-FALCON learning process under the same environment settings. For example, the exploration rate ϵ of a TD-FALCON system is decreasing across trials in favor of the exploration in the initial stage and the exploitation in the later stage. However, whether other parameters also need to be changed across the trials remains a question. Presently there are some developments [119, 120] in parameter selection in function approximation approaches, but they are basically limited to input variables. Addressing the above

issues, we will have to study the internal relationships between the TD-FALCON learning parameters, the reward schemes, the multi-agent environment settings (for example, the number of agents, the speed of each agent, the size of the domain, and so on) and the performance of the TD-FALCON system, establishing mathematical models for them and thus systematizing the selection of TD-FALCON parameters and reward scheme.

Appendix A

List of Publications

Journal Papers

1. Dan Xiao and Ah-Hwee Tan, "Self-Organizing Neural Architectures and Cooperative Learning in Multi-Agent Environment," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 37, no. 6, pp. 1567–1580, 2007.

2. Ah-Hwee Tan, Ning Lu, and Dan Xiao, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Transactions on Neural Networks*, vol. 9, no. 2, pp. 230–244, 2008.

3. Dan Xiao and Ah-Hwee Tan, "Cooperative reinforcement learning in topology-based multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, under review.

Conference Papers

1. Ah-Hwee Tan and Dan Xiao, "Self-organizing cognitive agents and reinforcement learning in multi-agent environment," in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'2005)*, Compiegne, France, 19-22 sep 2005, pp. 351-357.

2. Dan Xiao and Ah-Hwee Tan, "Cooperative cognitive agents and reinforcement learning in pursuit game," in *Proceedings of Third International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS'05)*, Singapore, 2005.

3. Dan Xiao and Ah-Hwee Tan, "Scaling up multi-agent reinforcement learning in complex domains," in *Proceedings of 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Sydney, 2008, pp. 326–329.

Bibliography

[1] V. Lesser, "Cooperative multiagent systems: A personal view of the state of the art," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 133–142, January 1999.

[2] N. R. Jennings, "Coordination techniques for distributed artificial intelligence," *Foundations of distributed artificial intelligence*, pp. 187–210, 1996.

[3] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," Tech. Rep., George Mason University, 2003, Technical Report GMU-CS-TR-2003-1.

[4] Y. Liu and S. Yao, "Research on the multi-agent model of autonomous distributed control systems," in *Proceedings of 31st International Conference on Technology of Object-Oriented Language and Systems*, Nanjing, China, 1999, pp. 331–335.

[5] H. P. Schwefel, "Advantages and disadvantages of evolutionary computation over other approaches, evolutionary computation 1: Basic algorithms and operators," 2000.

[6] M. Kaya and R. Alhajj, "Modular fuzzy-reinforcement learning approach with internal model capabilities for multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no. 2, pp. 1210–1223, 2004.

[7] M. Kaya and R. Alhajj, "Fuzzy olap association rules mining-based modular reinforcement learning approach for multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 2, pp. 326–338, 2005.

[8] A.-H. Tan and D. Xiao, "Self-organizing cognitive agents and reinforcement learning in multi-agent environment," in *Proceedings of 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'2005)*, Compiegne, France, 19–22 sep 2005, pp. 351–357.

[9] L. Busoniu, R. Babuska, and B. De Schutter, "Multi-agent reinforcement learning: A survey," in *Proceedings of 9th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Singapore, 2006, pp. 1–6.

[10] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: a critical survey," Tech. Rep., Stanford University, 2003.

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

[12] H. Kose, U. Tatlidede, C. Mericli, K. Kaplan, and H. L. Akin, "Q-learning based market-driven multi-agent collaboration in robot soccer," in *Proceedings of Turkish Symposium On Artificial Intelligence and Neural Networks*, Izmir, Turkey, 2004, pp. 219–228.

[13] V. Kononen, "Gradient descent for symmetric and asymmetric multiagent reinforcement learning," *Web Intelligence and Agent Systems: An International Journal (WIAS)*, vol. 3, no. 1, pp. 17–30, 2005.

[14] D.H. Ackley and M.L. Littman, "Generalizaionm and scaling in reinforcement learning," in *Advances in Neural Information Processing Systems 2*, 1990, pp. 550–557.

[15] R.S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*, Ph.D. thesis, University of Massachusetts, Amherst, MA, 1984.

[16] L.J. Lin, "Programming robots using reinforcement learning and teaching," in *Proceedings of 9th National Conference on Artificial Intelligence*, Anaheim, California, 1991, pp. 781–786.

[17] M. L. Littman, "Markov games as a framework for multiagent reinforcement learning," in *Proceedings of 11th international conference on Machine learning*, San Francisco, CA, 1994, pp. 157–163.

[18] M. Tan, "Multi-agent reinforcement learning: independent vs. cooperative agents," in *Proceedings of 10th International Conference on Machine Learning*, Amherst, MA, USA, 1993, pp. 330–337, Morgan Kaufmann.

[19] J. Hu and M. Wellman, "Multiagent reinforcement learning: theoretical framework and an algorithm," in *Proceedings of 15th International Conference on Machine Learning*, Madison, Wisconsin, USA, 1998, pp. 242–250.

[20] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. 1994, vol. 6, pp. 671–678, Morgan Kaufmann Publishers, Inc.

[21] J. Schneider, W. K. Wong, A. Moore, and M. Riedmiller, "Distributed value functions," in *Proceedings of 16th International Conference on Machine Learning*, San Francisco, CA, 1999, pp. 371–378, Morgan Kaufmann.

[22] Y. H. Chang, T. Ho, and L. P. Kaelbling, "Mobilized ad-hoc networks: a reinforcement learning approach," in *Proceedings of 2004 International Conference on Autonomic Computing*, New York, NY, 2004, pp. 240–247.

[23] J. Hu and M. Wellman, "Online learning about other agents in a dynamic multiagent system," in *Proceedings of 2nd International Conf. on Autonomous Agents*, 1998, pp. 239–246.

[24] 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 Transactions on Neural Networks*, vol. 9, no. 2, pp. 230–244, 2008.

[25] M. Mundhe and S. Sen, "Evaluating concurrent reinforcement learners," in *Proceedings of 4th International Conference on Multiagent Systems (ICMAS'00)*, Boston, MA, USA, 2000, pp. 421–422.

[26] S. Sen and M. Sekaran, "Individual learning of coordination knowledge," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 10, no. 3, pp. 333–356, 1998.

[27] S. Sen, M. Sekaran, and J. Hale, "Learning to coordinate without sharing information," in *Proceedings of 12th National Conference on Artificial Intelligence*, Seattle, Washington, USA, 1994, pp. 426–431.

[28] A.-H. Tan, "Self-organizing neural architecture for reinforcement learning," in *Proceedings of International Symposium on Neural Networks (ISNN'06), LNCS 3971, Chengdu, China*, 2006, pp. 470–475.

[29] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54–115, June 1987.

[30] G. A. Carpenter and S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, vol. 26, pp. 4919–4930, July 1987.

[31] D. Xiao and A.-H. Tan, "Self-organizing neural architectures and cooperative learning in multi-agent environment," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, vol. 37, no. 6, pp. 1567–1580, 2007.

[32] D. Xiao and A. H. Tan, "Cooperative cognitive agents and reinforcement learning in pursuit game," in *Proceedings of 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS'05)*, Singapore, 2005.

[33] E.F. Yang and D.B. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," Tech. Rep., Department of Computer Science, University of Essex, 2004, Technical Report CSM-404.

[34] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," in *Proceedings of 18th national conference on Artificial intelligence*, Menlo Park, CA, USA, 2002, pp. 326–331, American Association for Artificial Intelligence.

[35] G. Chalkiadakis and C. Boutilier, "Coordination in multiagent reinforcement learning: A bayesian approach," in *Proceedings of 2nd International Joint Conference on Autonomous Agents & Multiagent Systems*, 2003, pp. 709–716.

[36] D. Xiao and A.-H. Tan, "Cooperative reinforcement learning in topology-based multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, under review.

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

[38] M. Brenda, V. Jagannathan, and R. Dodhiawala, "On optimal cooperation of knowledge sources - an empirical investigation," Tech. Rep., Boeing Advanced Technology Center, 1986, Technical Report BCS-G2010-28.

[39] D. Xiao and A.-H. Tan, "Scaling up multi-agent reinforcement learning in complex domains," in *Proceedings of 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Sydney, 2008, pp. 326–329.

[40] M. A. Arbib, *The Handbook of Brain Theory and Neural Networks*, MIT Press, 2002.

[41] O. Azouaoui, A. Cherifi, R. Bensalem, and A. Farah, "Reinforcement learning based group navigation approach for multiple autonomous robotic system," in *Proceedings of 2005 IEEE International conference on Mechatronics and Automation*, Niagara Falls, Ont., Canada, 2005, vol. 3, pp. 1539–1544.

[42] M. Dubreuil, C. Gagne, and M. Parizeau, "Analysis of a master-slave architecture for distributed evolutionary computations," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 1, pp. 229–235, 2006.

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

[44] L.P. Kaelbling, M.L. Littman, and A.W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[45] D. Szer and F. Charpillet, "Improving coordination with communication in multi-agent reinforcement learning," in *Proceedings of 16th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 04*, INRIA, LORIA, Vandoeuvre-les-Nancy, France, 15–17 Nov. 2004, pp. 436–440.

[46] T. M. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[47] M. Riedmiller and H. Braun, "Rprop – a fast adaptive learning algorithm," Tech. Rep., Universitat Karlsruhe, 1992, Technical Report (Also Proc. of ISCIS VII).

[48] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, 1993, pp. 586–591.

[49] M. Riedmiller and H. Braun, "Rprop – description and implementation details," Tech. Rep., University of Karlsruhe, Karlsruhe, Germany, 1994.

[50] G. Tesauro, "Extending q-learning to general adaptive multi-agent systems," 2003, citeseer.ist.psu.edu/630001.html.

[51] J. Y. Choi and H. J. Park, "Use of neural networks in iterative learning control systems," *International Journal of Systems Science*, vol. 31, no. 10, pp. 1227–1239, 2000.

[52] A.-H. Tan, "FALCON: A fusion architecture for learning, cognition, and navigation," in *Proceedings of 2004 IEEE International Joint Conference on Neural Networks*, Budapest, Hungary, 2004, pp. 3297–3302.

[53] A. H. Tan and H. S. Soon, "Concept hierarchy memory model: A neural architecture for conceptual knowledge representation, learning, and commonsense reasoning," *International Journal of Neural Systems*, vol. 4, no. 2, pp. 93–124, 1996.

[54] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "ART 2-A: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 493–504, 1991.

[55] K.S. Hwang, S.W. Tan, M.C. Hsiao, and C.S. Wu, "Cooperative multiagent congestion control for high-speed networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 2, pp. 255–268, 2005.

[56] N. Ono and K. Fukomoto, "Multi-agent reinforcement learning: a modular approach," in *Proceedings of 2nd International Conference on Multi-Agent Systems*, 1996, pp. 252–258.

[57] L. Bull and T. C. Fogarty, "Evolving cooperative communicating classifier systems," in *Proceedings of 4th Annual Conference on Evolutionary Programming*, 1994, pp. 308–315.

[58] T. Balch, "Reward and diversity in multirobot foraging," in *Proceedings of the "Agents Learning about, from and with other Agents" Workshop*, Stockholm, Sweden, 1999.

[59] J. M. Vidal and E. H. Durfee, "The moving target function problem in multi-agent learning," in *Proceedings of 3rd International Conference on Multi-Agent Systems*, Paris, France, July 1998, pp. 317–324.

[60] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Advances in Complex Systems*, pp. 265–279, 2001.

[61] T. Balch, "Learning roles: Behavioural diversity in robot teams," Tech. Rep., Georgia Institute of Technology, 1997, Technical Report GIT-CC-97-12.

[62] M. Mataric, "Learning to behave socially," in *Proceedings of 3rd International Conference on Simulation of Adaptive Behaviour*. 1994, GIT–CC–97–12, pp. 453–462, MIT Press.

[63] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proceedings of the National Conference on Artificial Intelligence*, 1998, pp. 746–752.

[64] M. Bowling and M. Veloso, "An analysis of stochastic game theory for multiagent reinforcement learning," Tech. Rep., Computer Science Department, Carnegie Mellon University, 2000, Technical Report CMU-CS-00-165.

[65] R. Mukherjee and S. Sen, "Towards a pareto-optimal solution in general-sum games," in *Proceedings of 2nd International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003*, Melbourne, Victoria, Australia, 2003, pp. 153–160.

[66] A. Agogino and K. Tumer, "Reinforcement learning in large multiagent systems," in *AAMAS-05 Workshop on Coordination of Large Scale Multi-Agent Systems*, Utrecht, Netherlands, 2005.

[67] M. Lauer and M. Riedmiller, "Reinforcement learning for stochastic cooperative multi-agent systems," in *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004, vol. 3, pp. 1516–1517.

[68] T. Villmann and E. Merenyi, "Extensions and modifications of the kohonen-som and applications in remote sensing image analysis," *Self–Organizing Maps: Recent Advances and Applications*, pp. 121–145, 2001.

[69] A. J. Smith, "Applications of the self-organising map to reinforcement learning.," *Neural Networks*, vol. 15, no. 8–9, pp. 1107–1124, 2002.

[70] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning

of analog multidimensional maps," *IEEE Transactions on Neural Networks*, vol. 3, pp. 698–713, 1992.

[71] A.-H. Tan, "Adaptive resonance associative map," *Neural Networks*, vol. 8, no. 3, pp. 437–446, 1995.

[72] S. Ninomiya, *A hybrid Learning Approach Integrating Adaptive Resonance Theory and Reinforcement Learning for Computer Generated Agents*, Ph.D. thesis, University of Central Florida, 2002.

[73] 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 Networks*, vol. 4, pp. 759–771, 1991.

[74] B. Moore, "ART 1 and pattern clustering," in *Proceedings of 1988 Connectionist Models Summer School*, 1989, pp. 174–185.

[75] M. Borga and T. Carlsson, "A Survey of Current Techniques for Reinforcement Learning," Report LiTH-ISY-I-1391, Computer Vision Laboratory, SE-581 83 Linköping, Sweden, 1992.

[76] A. Pérez-Uribe, *Structure-Adaptable Digital Neural Networks*, Ph.D. thesis, Swiss Federal Institute of Technology-Lausanne, 2002.

[77] D. Gordan and D. Subramanian, "A cognitive model of learning to navigate," in *Proceedings of 19th Annual Conference of the Cognitive Science Society*, 1997, pp. 271–276.

[78] Y. Bai, H. Zhang, and Y. Hao, "The performance of the backpropagation algorithm with varying slope of the activation function," *Chaos, Solitons & Fractals*, vol. 40, no. 1, pp. 69–77, 2007.

[79] C.J.C.H. Watkins, *Learning from delayed rewards*, Ph.D. thesis, King's College, Cambridge, UK, 1989.

[80] A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis, "A new learning rates adaptation strategy for the resilient propagation algorithm," in *Processings, ESANN*, 2004, pp. 1–6.

[81] P. Stone, "Layered learning in multiagent systems," in *AAAI/IAAI*, 1997, p. 819.

[82] J. Denzinger and M. Fuchs, "Experiments in learning prototypical situations for variants of the pursuit game," in *Proceedings of 2nd International Conference on Multiagent Systems*, Kyoto, Japan, 1996, pp. 48–55.

[83] Jörg Denzinger and Alvin Schur, "On customizing evolutionary learning of agent behavior.," in *Proceedings of Canadian Conference on AI*, 2004, pp. 146–160.

[84] T. W. Sandholm and R. H. Crites, "Multiagent reinforcement leaning in the iterated prisoner's dilemma," *Biosystems*, vol. 37, pp. 147–166, 1995.

[85] S. Arai and K. Sycara, "Effective learning approach for planning and scheduling in multi-agent domain," in *Proceedings of 6th International Conference on Simulation of Adaptive Behaviour (From animals to animats 6)*, 2000, pp. 507–516.

[86] V. R. Lesser, D. D. Corkill, and E. H. Durfee, "An update on the distributed vehicle monitoring testbed," Tech. Rep., University of Massachusetts, Amherst, MA, USA, 1987.

[87] L. Nunes and E. Oliveira, "Learning from multiple sources," in *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-2004)*, 2004, vol. 3, pp. 1106–1113.

[88] L. Z. Varga, N. R. Jennings, and D. Cockburn, "Integrating intelligent systems into a cooperating community for electricity distribution management," *Expert Systems with Applications*, vol. 7, no. 4, pp. 563–579, 1994.

[89] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Some recent advances in network flows," *SIAM Review*, vol. 33, no. 2, pp. 175–219, June 1991.

[90] M. N. Ahmadabadi, M. Asadpour, and E. Nakano, "Cooperative q-learning: the knowledge sharing issue," *Advanced Robotics*, vol. 15, no. 8, pp. 815–832, 2001.

[91] K. Ito, A. Gofuku, Y. Imoto, and M. Takeshita, "A study of reinforcement learning with knowledge sharing for distributed autonomous system," in *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Okayama, Japan, 16–20 Jul. 2003, vol. 3, pp. 1120–1125.

[92] Z.-Q. Chen, A. T. Holle, B. M. E. Moret, J. Saia, and A. Boroujerdi, "Network routing models applied to aircraft routing problems," in *Proceedings of 27th conference on Winter simulation*, Arlington, Virginia, United States, 1995, pp. 1200–1206, IEEE Computer Society.

[93] R. C. Prior, R. L. Slavens, J. Trimarco, V. Akgun, E. G. Feitzinger, and C.-F. Hong, "Menlo worldwide forwarding optimizes its network routing," *Interfaces*, vol. 34, no. 1, pp. 26–38, 2004.

[94] H. N. Psaraftis, M. M. Solomon, T. L. Magnanti, and T.-U. Kim, "Routing and scheduling on a shoreline with release times," *Management Science*, vol. 36, no. 2, pp. 212–223, February 1990.

[95] A. Vannelli, "An interior point method for solving the global routing problem," in *Proceedings of 1989 IEEE Custom Integrated Circuits Conference*, San Diego, CA, USA, 15–18 May 1989, pp. 3.4/1–3.4/4.

[96] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi, "An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation," *Operations Research*, vol. 52, no. 5, pp. 723–738, September–October 2004.

[97] F. Chu, N. Labadi1, and C. Prins, "The periodic capacitated arc routing problem linear programming model, metaheuristic and lower bounds," *Journal of Systems Science and Systems Engineering*, vol. 13, no. 4, pp. 423–435, December 2004.

[98] J. A. Tomlin and J. S. Welch, "Formal optimization of some reduced linear programming problems," *Mathematical Programming*, pp. 232–240, October 1983.

[99] L. S. Grigor'eva, "A class of heuristic algorithms for the routing problem," *Journal of Mathematical Sciences*, vol. 73, no. 5, pp. 544–549, February 1995.

[100] K. Q. Zhu, K. C. Tan, and L. H. Lee, "Heuristics for vehicle routing problem with time," in *Proceedings of 6th International Symposium on Artificial Intelligence and Mathematics, AMAI 2000*, 2000.

[101] L.-J. Sun, X.-P. Hu, Y.-X. Li, J. Lu, and D.-L. Yang, "A heuristic algorithm and a system for vehicle routing with multiple destinations in embedded equipment," in *Proceedings of 7th International Conference on Mobile Business, 2008, ICMB '08*, Barcelona, 7–8 July 2008, pp. 1–8.

[102] B. Rathnasabapathy and P. Gmytrasiewicz, "Formalizing multi-agent pomdps in the context of network routing," AAAI Technical Report WS-02-12, Department of Computer Science, University of Illinois at Chicago, 2002.

[103] S.J. Han and S.B. Cho, "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 3, pp. 559–570, 2006.

[104] J. Venkateswaran, Y.-J. Son, A. T. Jones, and H.-S. J. Min, "A hybrid simulation approach to planning in a vmi supply chain," *International Journal of Simulation and Process Modelling*, vol. 2, no. 3–4, pp. 133–149, 2006.

[105] C. Almeder and M. Preusser, "A hybrid simulation optimization approach for supply chains," in *Proceedings of 6th EUROSIM Congress on Modelling and Simulation*, Ljubljana, Slovenia, 9–13 Sep. 2007, pp. 173–177.

[106] M. Munetomo, Y. Takai, and Y. Sato, "An intelligent network routing algorithm by a genetic algorithm," in *Proceedings of 4th International Conference on Neural Information Processing*, 1997, pp. 547–550.

[107] M. Al-Ghazal, A. El-Sayed, and H. Kelash, "Routing optimlzation using genetic algorithm in ad hoc networks," in *Proceedings of 2007 International Symposium on Signal Processing and Information Technology*, Giza, 15–18 December 2007, pp. 497–503.

[108] W.-C. Yeh, "An efficient memetic algorithm for the multi-stage supply chain network problem," *International Journal of Advanced Manufacturing Technology*, vol. 29, no. 7–8, pp. 803–813, 2006.

[109] M. Littman and J. Boyan, "A distributed reinforcement learning scheme for network routing," Tech. Rep., Robotics Institute, Pittsburgh, PA, USA, 1993, CMU–CS–93–165.

[110] U. Hiroyuki, Y. Takashi, I. Shigeya, T. Akinobu, and Q. Fei, "A distributed network routing algorithm using reinforcement learning method," *Transactions of the Institute of Electrical Engineers of Japan. C*, vol. 122–C, no. 6, pp. 922–927, February 2002.

[111] S. Khodayari and M.J. Yazdanpanah, "Network routing based on reinforcement learning in dynamically changing networks," in *Proceedings of 17th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 05*, Hong Kong, 16 Nov. 2005, pp. 362–366.

[112] T. van Tongeren, U. Kaymak, D. Naso, and E. van Asperen, "Q-learning in a competitive supply chain," in *Proceedings of IEEE International Conference on ISIC' 2007*, 2007, pp. 1211–1216.

[113] V. Stephan, K. Debes, H.-M. Gross, F. Wintrich, and H. Wintrich, "A reinforcement learning based neural multiagent system for controlof a combustion process," in *Proceedings of IEEE-INNS-ENNS International Joint Conference on IJCNN 2000*, Como, Italy, 2000, vol. 6, pp. 217–222.

[114] J. Bradley and G. Hayes, "Adapting reinforcement learning for computer games: Using group utility functions," in *Proceedings of IEEE Symposium on Computational Intelligence and Games, 2005*, Colchester, Essex, UK, 2005.

[115] D. H. Wolpert, K. Tumer, and J. Frank, "Using collective intelligence to route internet traffic," in *Proceedings of 1998 conference on Advances in neural information processing systems II*, Cambridge, MA, USA, 1999, pp. 952–958, MIT Press.

[116] D. Subramanian, P. Druschel, and J. Chen, "Ants and reinforcement learning: A case study in routing in dynamic networks," in *Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI–97)*. 1997, pp. 832–838, Morgan Kaufmann.

[117] M. Benhamadou, "On the simplex algorithm 'revised form'," *Advances in Engineering Software*, vol. 33, no. 11–12, pp. 769–777, 2002.

[118] G. B. Dantzig, A. Orden, and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality restraints," *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[119] L. J. Herrera, H. Pomares, I. Rojas, M. Verleysen, and A. Guilen, "Effective input variable selection for function approximation," in *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 2006), Part I*, Athens, Greece, September 2006, vol. LNCS 4131, pp. 41–50.

[120] J. Tikka, "Input selection for radial basis function networks by constrained optimization," in *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN 2007), Part I*, Porto, Portugal, September 2007, vol. 4668, pp. 239–248.