10-2011

Cooperative reinforcement learning in topology-based multi-agent systems

Dan XIAO

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

Citation

XIAO, Dan and TAN, Ah-hwee. Cooperative reinforcement learning in topology-based multi-agent systems. (2011). *Autonomous Agents and Multi-Agent Systems*. 26, (1), 86-119. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/5242

Auton Agent Multi-Agent Syst (2013) 26:86–119
DOI 10.1007/s10458-011-9183-4

Cooperative reinforcement learning in topology-based multi-agent systems

Dan Xiao · Ah-Hwee Tan

Published online: 7 October 2011

Abstract Topology-based multi-agent systems (TMAS), wherein agents interact with one another according to their spatial relationship in a network, are well suited for problems with topological constraints. In a TMAS system, however, each agent may have a different state space, which can be rather large. Consequently, traditional approaches to multi-agent cooperative learning may not be able to scale up with the complexity of the network topology. In this paper, we propose a cooperative learning strategy, under which autonomous agents are assembled in a binary tree formation (BTF). By constraining the interaction between agents, we effectively unify the state space of individual agents and enable policy sharing across agents. Our complexity analysis indicates that multi-agent systems with the BTF have a much smaller state space and a higher level of flexibility, compared with the general form of n-ary ($n > 2$) tree formation. We have applied the proposed cooperative learning strategy to a class of reinforcement learning agents known as temporal difference-fusion architecture for learning and cognition (TD-FALCON). Comparative experiments based on a generic network routing problem, which is a typical TMAS domain, show that the TD-FALCON BTF teams outperform alternative methods, including TD-FALCON teams in single agent and n-ary tree formation, a Q-learning method based on the table lookup mechanism, as well as a classical linear programming algorithm. Our study further shows that TD-FALCON BTF can adapt and function well under various scales of network complexity and traffic volume in TMAS domains.

Keywords Topology-based multi-agent systems · Cooperative learning · Reinforcement learning · Binary tree formation · Policy sharing

D. Xiao (✉) · A.-H. Tan
School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore
e-mail: XIAO1@e.ntu.edu.sg; Daniel.DXiao@gmail.com

A.-H. Tan
e-mail: asahtan@ntu.edu.sg

1 Introduction

Traditional multi-agent systems [1,2] assume that all agents can interact freely with each other and do not consider specific relations among the agents. In this paper, we study a class of topology-based multi-agent systems (TMAS), wherein each individual agent interacts only with its neighbors according to their spatial relationship. Though specialized in structure, TMAS systems are well suited for a wide range of problems with topological constraints, such as distributed vehicle monitoring [3,4], network management and routing [5,6], and electricity distribution management (EDM) [7,8].

Typically, a TMAS domain involves transmitting (or distributing) a number of objects from one or multiple point(s) (starting point) to another one or multiple point(s) (destination) through a set of networked intermediate nodes (routing nodes). A direct solution to the TMAS system is the so called *single agent strategy*, which employs one autonomous agent for each routing node. Each agent learns an optimal policy through the sensory and reinforcement feedback from their neighbors and the environment. However, in a TMAS system, cooperative reinforcement learning (RL) may pose many challenges. First of all, the agents have to handle the added spatial constraints. Having topological relationship with a varying number of neighbors, each agent has to deal with a different state space. In addition, as the number of agents increases, the agents have to adapt to the increasing complexity of the spatial relationship.

In this paper, we propose a binary tree formation (BTF) strategy, under which a team of agents, organized in BTF, is deployed for each routing node. As each agent now interacts with its neighboring agents in a fixed (binary) topology, the state space does not increase with the complexity of the network topology in a TMAS domain. The uniform state space representation also enables the agents to perform knowledge sharing, thus boosting the learning and operational efficiency.

To illustrate the BTF strategy, we adopt a self-organizing neural model called temporal difference-fusion architecture for learning and cognition (TD-FALCON) [9,10] as the RL agent. FALCON is a 3-channel fusion adaptive resonance theory (fusion ART) [11,12] architecture that learns action and value policies using RL across the sensory, action and feedback channels. By incorporating temporal difference (TD) methods to handle delayed evaluative rewards, the TD-FALCON model has shown to exhibit competitive learning capabilities, compared with gradient descent based RL systems [13–15]. Based on TD-FALCON, we propose the TD-FALCON BTF system, in which a number of TD-FALCON agents are grouped into a BTF, for cooperative learning in a TMAS domain.

In the most general case, TD-FALCON agents can be deployed at each routing node in an 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 have the disadvantages of increasing the size of the state and action spaces significantly. In addition, it may lack the flexibility of the BTF in handling a varying type of the network topology.

We have conducted comparative experiments to evaluate the performance of the TD-FALCON BTF algorithm on a standard network routing (NR) problem, which is a typical TMAS domain. Comparing with the TD-FALCON teams using the single agent and n-ary ($n > 2$) tree formations, a Q-learning algorithm based on the table lookup mechanism, as well as a classical linear programming algorithm, TD-FALCON BTF exhibits superior performance in terms of the success rate, the number of hops, the number of cognitive nodes learned, and the running time. In addition, we present a detailed study, to investigate the scalability of the TD-FALCON BTF system under a varying level of network complexity and traffic volume.

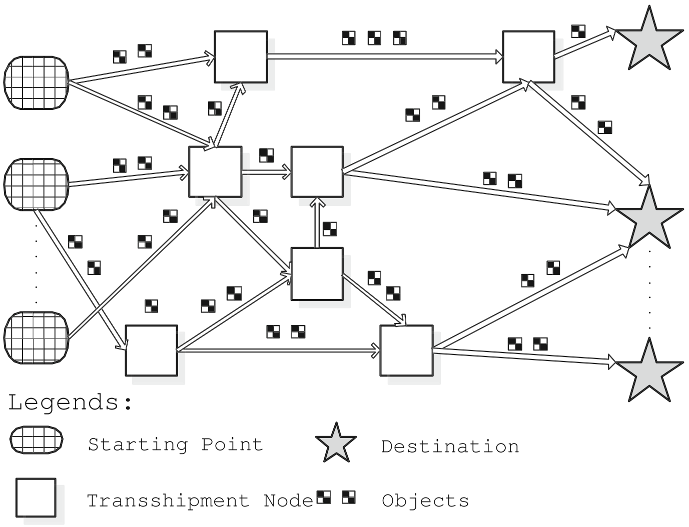

Fig. 1 A TMAS domain

The rest of the paper is organized as follows. Section 2 describes the TMAS problems and the related work. Section 3 introduces the TD-FALCON-based single agent approach. Section 4 presents the TD-FALCON BTF strategy and makes comparison with the TD-FALCON n-ary tree formation. Section 5 reports the experiments and results in the NR problem. Section 6 concludes and highlights the future work.

2 The TMAS problem domain

In this section, we first present the TMAS problem in a general form. Following that, examples and related works of TMAS are reviewed.

2.1 Problem definition

As shown in Fig. 1, in a TMAS task, a number of objects need to be sent from the p $(p > 0)$ starting points $S = \{S_1, S_2, \ldots, S_p\}$ to the q $(q > 0)$ destinations $D = \{D_1, D_2, \ldots, D_q\}$, 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 TMAS 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 all the starting points, the destinations and the transshipment nodes in the TMAS system, so that $V = S \cup D \cup T$. An arc $e = (x, y)$ refers to a one-way transfer path from node x to node y; y is called the **head** and x is called the **tail** of the arc e; y is named as a **successor** of x and x is named as a **predecessor** of y. We define a node R as a *routing node* if $deg^+(R) > 1$.[1] A routing node needs to route and send an object among a few alternative transfer paths.

[1] In a directed graph, for a node v, the number of head endpoints adjacent to v is called the **indegree** of the node v, denoted $deg^-(v)$; the number of tail endpoints adjacent to v is called the **outdegree** of the node v, denoted $deg^+(v)$.

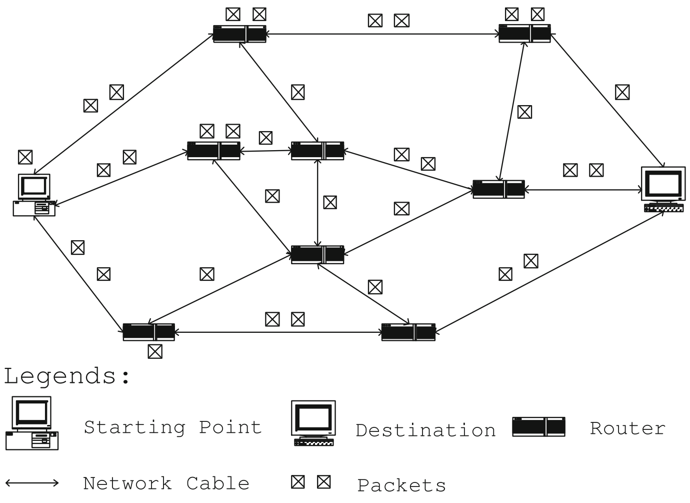

Fig. 2 A network routing domain

Capacities and transfer speeds among transshipment nodes can be different. Each transshipment node is equipped with a queue of specific size to store the received objects. A traffic jam may occur if too many objects are queued in a transshipment node. A *hop* is a routing cycle in which an object is transferred between two nodes, and different transshipment nodes can transfer objects concurrently within a hop. A TMAS task is deemed as a $success$ if all objects can be transferred to the destination(s) within the specified number of hops. The TMAS system will also try to finish transferring those objects as soon as possible. If any traffic jam occurs or time is up, the TMAS task is deemed as a $failure$.

2.2 Examples of TMAS problems

TMAS is applicable to many problem domains in real world. We review below some well-known multi-agent problems, namely the NR problem [16,17], the EDM [7,8] system, the multi-machine scheduling [18] and the robot soccer game [19,20]. Other problems, of which TMAS is applicable, include air traffic control [21], distributed vehicle monitoring [3,4], distributed medical care [22], meeting scheduling [23,24], and supply chain management [25], as reviewed in the related work section.

A typical TMAS domain is the NR problem. The task of the NR system is to transfer a given number of packets from a starting point to a chosen destination through a distributive network composed of a number of routers, as shown in Fig. 2. We can see that the network connections among those routers correspond directly to their spatial relationships.

The EDM problem is to transport and distribute a certain amount of electric power from one or multiple power stations to a number of residences through networked transformers. An example of the EDM problem is shown in Fig. 3. It can be noticed that the spatial relationships between an electric generator and a transformer, between two transformers, as well as between a transformer and a residence are implemented by those high tension power lines laid between them.

As shown in Fig. 4, the multi-machine scheduling (MMS) problem involves a number of machines performing a task collectively to manufacture a product out of various raw materials. The manufacturing process of each product is composed of r production steps,

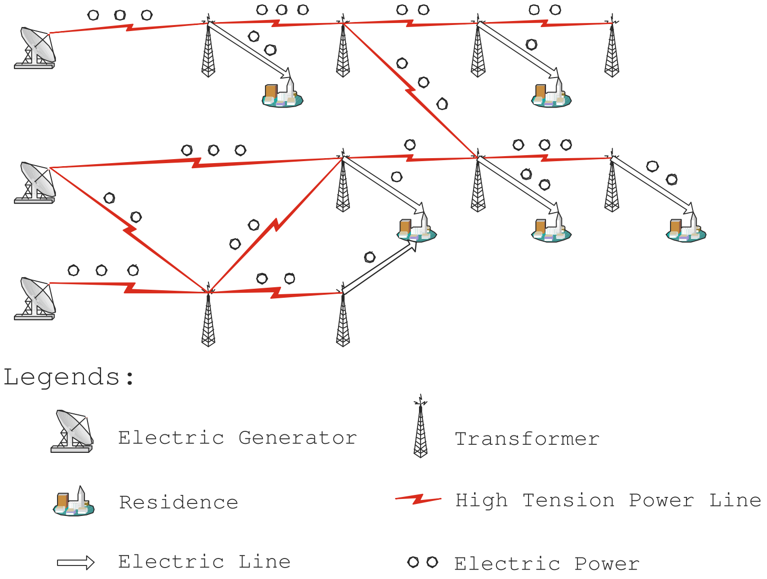

Fig. 3 An EDM system

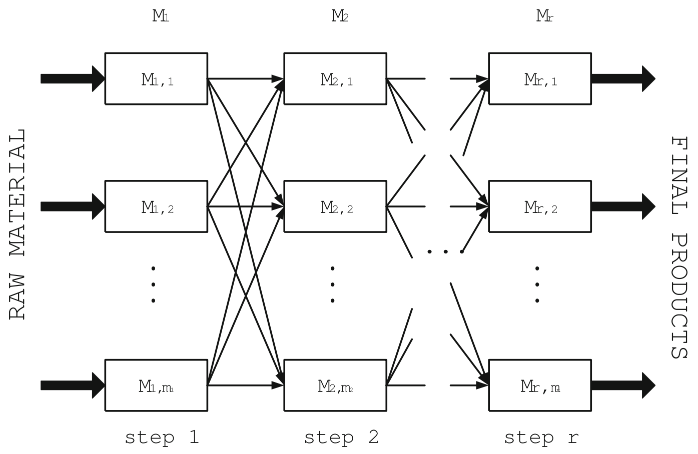

Fig. 4 A multi-machine scheduling system

where step i must be fulfilled before step $i+1$ can start for all $i \in [1, r-1]$. In each step M_i $(1 \le i \le r)$, there are m_i machines in service. If a product process is proceeding at the machine $M_{i,j}$ of the step i $(1 \le i \le r-1$ and $1 \le j \le m_i)$, the system must decide which machine the raw material should be forwarded to at the next step. As a result, the machine $M_{i,j}$ and each machine $M_{i+1,k}$ $(1 \le k \le m_{i+1})$ form a strong spatial relationship.

A more atypical example of the TMAS domain is the robot soccer problem. In a robot soccer game, an episode of an attacking task from a soccer team requires the passing of the ball from the goalkeeper to a forward, through the lines of the defenders and the midfielders, as shown in Fig. 5. Though the player positions are not fixed, a soccer team typically plays according to a specific formation. The passing paths of the ball thus constitute the spatial

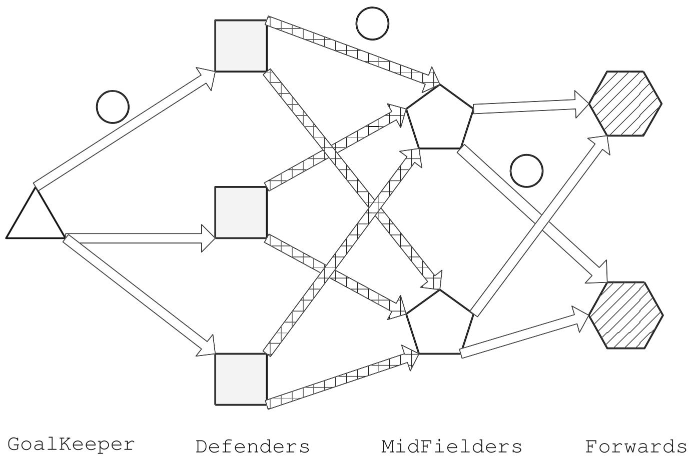

Fig. 5 An robot soccer game

relationships between the goalkeeper and defenders, defenders and midfielders, as well as midfielders and forwards. As these spatial relationships are generally stable during a match, the robot soccer game can be represented as a form of TMAS domain.

2.3 Related work

Operations research (OR) methods, in particular linear programming (LP), have been used extensively in many TMAS domains. For example, Vannelli [26] proposes a solution using the Interior Point Method, in which an LP projection process starts with a point known as the initial guess inside the n-dimensional linearly constrained space and constructs ellipsoids inside this feasible region, for solving a very large scale integrated circuit layout problem, i.e., the *global routing* problem. Roling and Visser [27] propose a mixed-integer linear programming [28], where only some of the unknown variables in LP equations are required to be integers (see Sect. 5.3 for the details), to implement a surface traffic automation system helping controllers to better coordinate surface traffic movements related to arrival and departure traffic in an airport. LeBlanc et al. [29] develop a spreadsheet linear-programming model for planning shipments of finished goods between vendors, manufacturing plants, warehouses, and customers to minimize overall cost for Nu-kote International, which is the largest independent manufacturer and distributor of aftermarket imaging supplies for home and office printing devices. The graphical method [30,31] has been widely used to solve the linear programming problem involving two variables x and y. There are usually two major steps in the graphical method: (a) Determine the solution space that defines the feasible solution; (b) Determine the optimal solution from the feasible region. Generally, the linear programming methods have shown the best possible usage of available productive resources and improvement in the quality of decision-making. However, the LP methods have two prominent disadvantages: (1) They may produce non-optimal solutions, because some variables are ignored; and (2) The graphical method of LP is restricted to two variables.

Heuristic methods have been found effective in some TMAS cases. Szozda et al. [32] use heuristic methods of forecasting in the planning and forecasting area of supply chain activity. Zhu et al. [33] explore various third generation heuristic methods, such as Interchange, Simulated Annealing, and Tabu Search to solve the vehicle routing problem with time windows, wherein the objective is to serve a number of customers within predefined time windows at minimum cost, without violating the capacity and total trip time constraints for each vehicle. Sun et al. [34] present a heuristic algorithm, based on the consideration of road conditions and special direction roads (one-way street, ban of turn in crossing, etc.), appropriate for vehicle routing problem with multiple destinations. Although heuristic methods may be effective in some ad-hoc cases, there is no way to prove the optimality of a heuristic algorithm. It may be correct, but may not be guaranteed to produce an optimal solution [35]. Heuristic methods are typically used when there is no known method to find an optimal solution or under the given constraints (of time, space etc.), because many of them are based on experiences, intuitive judgments or common sense [36].

More sophisticated optimization techniques based on Markov decision processes (MDPs) and partially observable MDPs (POMDPs) can be extended to uncertain environments. Rathnasabapathy and Gmytrasiewicz [37] use POMDPs as a basic framework to formalize NR as a multi-agent decision problem. Schurr [38] presents a personal assistant meeting scheduling domain using POMDPs, where the location of a meeting, number of attendees etc. would be a part of the state. Han [39] proposes the localized adaptive QoS routing scheme using POMDPs. However, those optimization techniques cannot scale well to complex TMAS environments.

Consequently, the use of genetic algorithms has been proposed to address the shortcoming of the POMDP approaches in TMAS. Munetomo et al. [40] propose an adaptive routing algorithm using genetic operators to realize an intelligent routing which directly observes communication latency of the routes. Lesser et al. [3] use a genetic algorithm to implement a novel generic architecture for the distributed vehicle monitoring model. Hu and Paolo [41] apply genetic algorithms to tackle the aircraft arrival sequencing and scheduling issue, which is a major issue in the daily air traffic control operations. However, those generic algorithms cannot overcome the drawback of slow convergence [42], as they do not exploit local information [43], which is useful to the convergence.

Reinforcement learning techniques based on the single agent model have also been proposed. Littman and Boyan [44] provide a self-adjusting algorithm for packet routing, in which a RL module is embedded into each node of a switching network. Baek et al. [45] propose an adaptive inventory control model for a supply chain consisting of one supplier and multiple retailers with non-stationary customer demands. They use a RL technique, namely action-reward based learning, to enable the control parameter to adaptively change as customer demand pattern changes. Adaptive agents in the distributed vehicle monitoring testbed set by Wan and Braspenning [46] use the theory of RL to find an optimal strategy by performing trials. However, as Leopold et al. [47] point out, the traditional RL techniques have two drawbacks: (1) the training process of an agent takes a lot of time, because it has to go through many rounds of trial-and-error methods and the agent typically has to learn from scratch; and (2) the traditional RL approaches are weak at the generalization of experiences, and thus the agents may fail to act appropriately in unfamiliar environments. Additionally, the state-of-the-art RL methods have been found to perform poorly in large scale complex environments [48,49].

3 Single TD-FALCON approach

In view of the limitations of the existing RL methods, we have proposed to use a self-organizing neural model as the learning agents in our system. By inheriting the dynamics of the

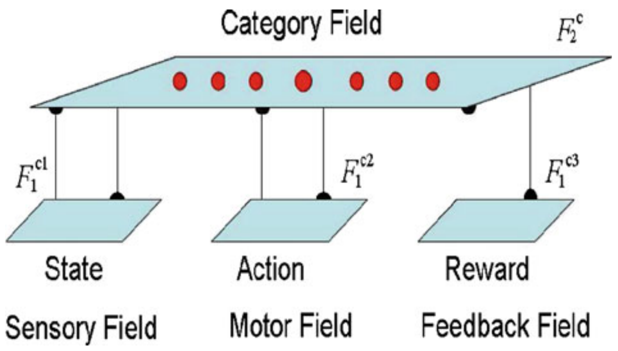

Fig. 6 The FALCON architecture

ART network, TD-FALCON is able to perform the online and incremental RL in a real-time environment. In addition, with the cooperative strategies resulting in reduced state space and high scalability, TD-FALCON systems have been shown to perform well in many multi-agent environments, with a varying level of complexity [13–15,50].

A straightforward solution to the TMAS problem is to deploy one RL agent for each routing node. This approach can be easily implemented and is commonly adopted [5,51,52]. 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 approach based on the TD-FALCON algorithm in this section.

3.1 FALCON architecture

FALCON is an extension of self-organizing neural networks called ART networks [11,12] for learning multi-modal pattern mappings across multiple input channels. Through a unique code stabilizing and dynamic network expansion mechanism, ART models are capable of learning that will do multi-dimensional mappings of input patterns in an online and incremental manner. They are thus suitable mechanisms for autonomous systems to learn value functions.

For RL, FALCON makes use of a 3-channel architecture (Fig. 6), consisting of a sensory field F_1^{c1} for representing the current state, an action field F_1^{c2} for representing the available actions, a reward field F_1^{c3} for representing the values of the feedback received from the environment, and a cognitive field F_2^c. In response to a continual stream of incoming patterns, FALCON learns cognitive nodes, each encodes a relation across the pattern in the three input channels. We describe how FALCON can be used to predict and learn value functions for RL below.

Input vectors Let $\mathbf{S} = (s_1, s_2, \ldots, s_n)$ denote the state vector, where s_i indicates the sensory input i. Let $\mathbf{A} = (a_1, a_2, \ldots, a_m)$ denote the action vector, where a_i indicates a possible action i. Let $\mathbf{R} = (r, \bar{r})$ denote the reward vector, where $r \in [0, 1]$ and $\bar{r}$ (the complement of r) is given by $\bar{r} = 1 - r$. Complement coding serves to normalize the magnitude of the input vectors and has been found effective in ART systems in preventing the code proliferation problem [53].

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

Weight vectors Let $\mathbf{w}_j^{ck}$ denote the weight vector (also known as weight template or template 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.
Parameters The FALCON's dynamics is determined by choice parameters $\alpha^{ck} > 0$ for $k = 1$ to 3; learning rate parameters $\beta^{ck} \in [0, 1]$ for $k = 1$ to 3; contribution parameters $\gamma^{ck} \in [0, 1]$ for $k = 1$ to 3, where $\sum_{k=1}^{3} \gamma^{ck} = 1$; and vigilance parameters $\rho^{ck} \in [0, 1]$ for $k = 1$ to 3.

The dynamics of FALCON for predicting and learning value functions, based on fuzzy ART operations [54], is described in Sects. 3.1.1 and 3.1.2 respectively.

3.1.1 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:

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

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$, for vectors $\mathbf{p}$ and $\mathbf{q}$.
Code competition A code competition process follows under which the F_2^c node with the highest choice function value is identified. The system is said to make a choice when at most one F_2^c node can become active after code competition. The winner is indexed at J where

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

When a category choice is made at node J, $y_J^c = 1$, and $y_j^c = 0$ for all $j \neq J$. This indicates a winner-take-all strategy.
Activity readout The chosen F_2^c node J performs a readout of its weight vectors to the input fields F_1^{ck} as

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

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

3.1.2 Learning

For learning value functions, the state, action, and reward vectors are presented simultaneously 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 Sect. 3.1.1) to select a winner J based on the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$, and $\mathbf{x}^{c3}$. To complete the learning process, template matching and template learning are performed as described below.

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

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

When a resonance occurs, the template learning ensues, as defined below. If any of the vigilance constraints (Eq. 4) is violated, mismatch reset occurs in which the value of the choice function T_J^c is reset to -1 during the input presentation. The search process then continues to select another F_2^c node J until a resonance is achieved.

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

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

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

Node creation Our implementation of FALCON always maintains ONE uncommitted node in the F_2^c field. 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 TD-FALCON

For learning from delayed evaluative feedback, FALCON was extended to temporal difference-FALCON (TD-FALCON) [10,55], that incorporates temporal difference learning rules for estimating future cumulative reward values. As summarized in Table 1, TD-FALCON operates in a general sense-act-learn cycle. 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. Due to the space constraints, the key steps of the TD-FALCON algorithm are briefly described in the following sections. Please refer to [14] for the detailed description and algorithm.

3.2.1 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 $\mathbf{x}^{c3} = (1, 1)$.

Table 1 Generic dynamics of TD-FALCON

1.	Initialize the FALCON network
2.	Given the current state s, for each available action a in the action set $\mathcal{A}$, predict the value of the action $Q(s, a)$ by presenting the corresponding state and action vectors **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 $\text{s} = \text{s}'$
8.	Repeat from Step 2 until s is a terminal state

With the activity vector values, the system performs code activation and code competition as described in Sect. 3.1.1. 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

$$\mathbf{x}^{c3} = \mathbf{x}^{c3(\text{old})} \wedge \mathbf{w}_J^{c3}. \tag{6}$$

Note that the constraint of $\sum_i x_i^{c3} = 1$ may not hold here after learning, even with complemented coded input patterns. The Q-value of performing the action a in the state s is then given by

$$Q(s, a) = \frac{x_1^{c3}}{\sum_i x_i^{c3}}. \tag{7}$$

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

3.2.2 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 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 RL agents is to explore the environment. If an agent keeps selecting the action that it believes to be optimal, 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 actions believed to be the best, and *exploration*, i.e., trying out other seemingly inferior and less familiar actions. The ϵ-greedy policy, designed to achieve a balance between exploration and exploitation, is presented below.

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 [56]. 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 decaying ϵ-greedy policy is thus proposed to gradually reduce the value of ϵ linearly over time. The decaying 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.

3.2.3 Value function estimation

One key component of TD-FALCON is the iterative estimation of the value function $Q(s, a)$ using a temporal difference equation $\Delta Q(s, a) = \alpha TD_{err}$, where $\alpha \in [0, \frac{1}{2}]$ is the learning parameter and TD_{err} is a function of the current Q-value predicted by FALCON and the Q-value newly computed by the TD formula. Using the Q-learning rule, the temporal error term is computed by

$$TD_{err} = r + \gamma \max_{a'} Q(s', a') - Q(s, a), \tag{8}$$

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'.

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

$$Q(s, a) = \begin{cases} 1 & \text{if } Q(s, a) > 1 \\ 0 & \text{if } Q(s, a) < 0 \\ Q(s, a) & \text{otherwise} \end{cases} \tag{9}$$

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 [10] is given by

$$\Delta Q(s, a) = \alpha TD_{err} \left(1 - Q\left(s, a\right)\right). \tag{10}$$

By introducing the scaling term $1 - Q(s, a)$, the adjustment of Q-values will be self-scaled so that they will not increase 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 [10].

From Eq. 8, the Q-value will increase with the reward r. As a result, the favorable actions can cause Q-values to converge to 1, and on the contrary, the unfavorable actions can make Q-values converge to 0. The Q-values can provide useful information to differentiate favorable and unfavorable actions.

3.3 Sensory representation and reward schema

In a TMAS 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 objects of each successor, as described below.

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

Queue volume is defined as the number of objects that a transshipment node can queue. For input normalization, we use the proportion of the queue volume C with respect 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 objects counts the number of objects that are queued at the successor. For normalization, we use the reciprocal of the number of the queued objects as a state input. Mathematically, the sensory input is denoted as $S_q = \frac{1}{1+P}$, where P is the number of queued objects. For example, if there are four objects queued in a transshipment node, the relevant sensory input is 0.2. This parameter is to be used in conjunction with the queue volume to reflect the current status of each successor.

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

$$S = (S_t, S_c, S_q). \tag{11}$$

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 as

$$A = \{a_0, a_1, \ldots, a_{m-1}\}, \tag{12}$$

where $a_I = 1$ if a_I corresponds to the action of transferring an object 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 $N = |T| \cdot |C| \cdot |P|$. If a routing node R has m successors, the overall size of the state space of R is

$$N^m. \tag{13}$$

A scalability problem occurs as the overall size of the state space of R increases with m exponentially.

As mentioned earlier, a major problem of the single TD-FALCON approach to the TMAS problem is the different state spaces required across the various routing nodes. Assume that a routing node R_G has g successors and another routing node R_H has h successors. On the condition of $g \neq h$, the TD-FALCON agents installed in the routing nodes R_G and R_H typically have a different architecture and thus knowledge sharing is impossible. As a result, the number of cognitive nodes may increase significantly.

4 TD-FALCON binary tree formation 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 BTF. For example, if the routing node R is to route and send objects 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 Fig. 7.

The working mechanism of a n-level ($n \geq 1$) TD-FALCON BTF is described in Tables 2 and 3. For the ease of description, we include the successor transshipment nodes as the leaf nodes of the binary tree. In the BTF, 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

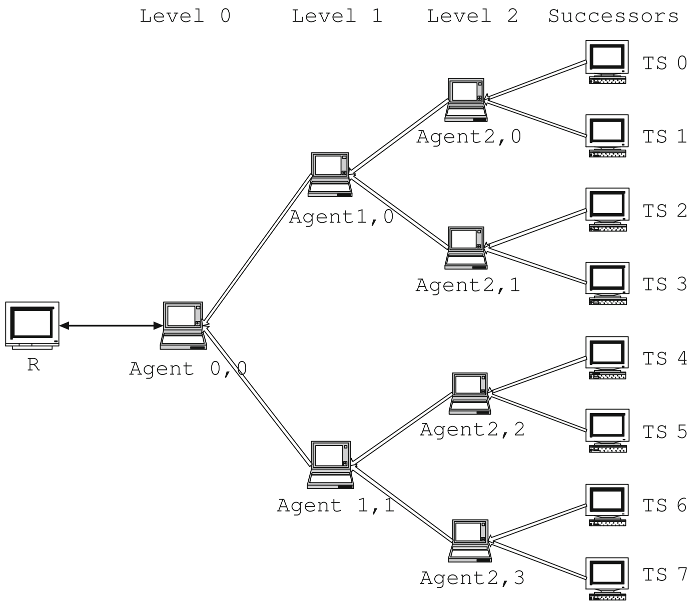

Fig. 7 A TD-FALCON binary tree formation with eight successor transshipment nodes

Table 2 The operational cycle of a branch-node agent $Agent_{p,q}$ in TD-FALCON BTF

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 an object 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, the action $a_{p+1,j}$, and the reward r
– $Agent_{p,q}$ transfers the reward r to $Agent_{p+1,j}$

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 Fig. 7 to demonstrate the TD-FALCON BTF working mechanism as follows.

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 an object 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 $Agent_{1,1}$.

Table 3 The operational cycle of the root-node agent $Agent_{0,0}$ 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 $a_{1,i}$ $(0 \le i \le 1)$ is the action of sending an object to the successor recommended from $Agent_{1,i}$, $Agent_{0,0}$ selects an action $a_{1,j}$ $(0 \le j \le 1)$
3.	After sending the object to TS_k $(0 \le k < 2^{n-1})$, 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, the action $a_{1,j}$, and the reward r

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 deployed at 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 learned and used by the various agents concurrently. In our simulation system, all the transshipment nodes are simulated by software running on the same computer. Thus knowledge sharing among them can be implemented easily by using the static variable mechanism in Object Oriented programming languages. However, in a physical NR environment, wherein the transshipment nodes are separate entities, knowledge sharing across different transshipment nodes may not be trivial and some form of communications will be required. For example, a "data center" may need to be established to communicate with the transshipment nodes and store the cognitive nodes that are shared by all agents.

4.1 Reward scheme of TD-FALCON BTF

We adopt a hybrid reward scheme [1] in the TD-FALCON BTF algorithm. For the local rewards, we use the expected transfer time (ETT) of an object as a measure, which is the number of hops that an object is expected to be transferred out from a transshipment node. Suppose that an object 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 P objects in queue. Then there are two possible situations: (1) if $P < C$, TS_j is not jammed, the ETT of TS_j is given by $ETT_j = T(P + 1)$; (2) if $P \ge C$, 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, such as 999. The rationale of using ETT as the local reward is that an object should be transferred to a successor transshipment node with the highest transfer speed.

As for the global rewards, we use the overall queued objects (OQO) to indicate the total number of objects queued in all transshipment nodes. Assume that there are altogether n transshipment nodes, where the mth transshipment node is marked as TS_m $(0 \le m \le n - 1)$ and the number of objects queued in TS_m is marked as Q_m. The OQO of the entire TMAS system can then be computed as $OQO = \sum_{m=0}^{n-1} Q_m$. Using OQO as the global reward is to transfer an object to a successor transshipment node with the lightest load.

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

$$R = \frac{1}{ETT_j(1 + OQO)}. \tag{14}$$

Using the hybrid reward, a routing node makes a balance between the requirement of sending objects in a short time and the minimization of traffic jam in the entire TMAS domain, by distributing objects evenly among the successors.

4.2 Complexity analysis of TD-FALCON BTF

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

$$S = m - 1. \tag{15}$$

It is noticed that all TD-FALCON agents in a BTF have a uniform state space (N^2), action space (two choices) and reward scheme. They are thus fully symmetrical in terms of both architecture and learning algorithms. Adopting the policy sharing method proposed by Tan [57], all those TD-FALCON agents in the binary tree are able to share the same set of cognitive nodes. 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 same TD-FALCON network configuration.

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, because there are only $\log_2 m$ levels in a m-node binary tree. The routing selection process can be conducted concurrently by 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, as long as there are enough processors (CPUs) operating in parallel.

4.3 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 an n-ary tree (where $n > 2$) for a routing node. Having a topological symmetry, an n-ary tree can also implement the policy sharing mechanism among all TD-FALCON agents. Therefore the estimated number of the cognitive nodes is in the same order as that of the TD-FALCON BTF.

Lemma *An 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 BTF.*

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 $U = \log_n m$, and then the total number of nodes in the n-ary tree is

$$S = 1 + n + n^2 + \cdots + n^{U-1} = \frac{n^U - 1}{n - 1} = \frac{m - 1}{n - 1}. \tag{16}$$

If we take $n = 2$, the total number of nodes is $m - 1$. This reduces to Eq. 15. □

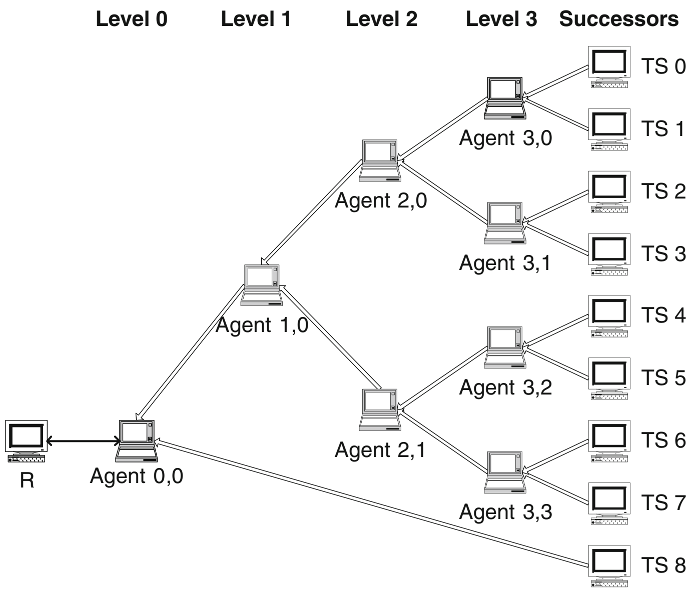

Fig. 8 A TD-FALCON binary tree formation for a routing node with nine successors

Equation 16 highlights a trend that with the growth of n, the number of TD-FALCON nodes in an n-ary tree is decreased in an inversely proportional manner, meaning that an n-ary tree TD-FALCON structure has the cost of $O\left(\frac{1}{n}\right)$ in the number of nodes.

In addition to having fewer TD-FALCON agents in construction, an 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 BTF ($log_2(m)$). Nevertheless, such an advantage can be minimized if there are enough CPUs/cores operating in parallel.

An n-ary tree structure has two disadvantages. Firstly, if the number (m) of successors of a routing node is not in the power of 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 would degrade the performance of such a system, because those "dummy" values constitute to "noises" in the sensory inputs and may interfere with other critical input values. On the contrary, a binary tree can keep the topological symmetry in a routing node with any number of successors. For example, we can still build a binary tree for a routing node with nine successors, as shown in Fig. 8.

The second disadvantage of an n-ary tree is the significant increment in state and action spaces. The number of actions that a TD-FALCON agent can select increases from 2 in a binary tree to n in an n-ary tree, so the increment in the action space is linear. According to Eq. 13, the state space exponentially increases from N^2 in a binary tree to N^n in an n-ary tree.

Considering a routing node with m successors, the time and space complexities of TD-FALCON teams in single agent, BTF, and n-ary tree formation are summarized in Table 4.

Table 4 Comparing the time and space complexities of TD-FALCON teams in single agent, binary tree formation, and n-ary tree formation

Complexity	Binary tree	n-ary tree	Single TD-FALCON
Number of cognitive nodes	$O(D^2)$	$O(D^n)$	$O(D^m)$
Delay time	$O(log_2(m))$	$O(log_n(m))$	$O(1)$
Size of state space	$O(D^2)$	$O(D^n)$	$O(D^m)$
Size of action space	$O(1)$	$O(n)$	$O(m)$

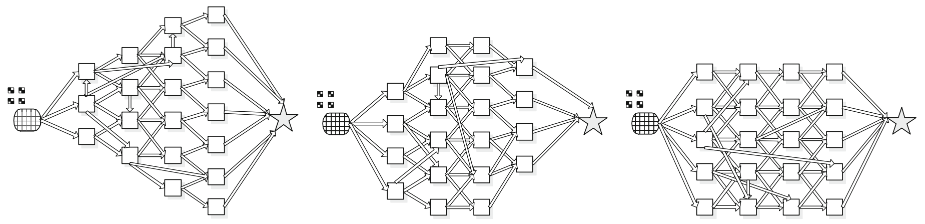

Fig. 9 Samples of network layout used in the experiments

D represents the dimension of the state vector received from one successor. m is the average number of successor transshipment nodes of a routing node.

5 Experimental results

In this section, we perform a series of experiments in a generic NR task to compare the performance of TD-FALCON teams in various configurations, with the performance of a Q-learning method based on the table lookup mechanism and that of a classical linear programming (LP) method. We also report experiments, where the scalability of the TD-FALCON BTF algorithm is tested in various network and traffic conditions.

5.1 Comparing TD-FALCON teams in various formations

In this group of experiments, we compare the performance of TD-FALCON teams in the single agent, binary tree, trinary tree, and 5-ary tree formations, in transferring 50 packets over a network of 20 transshipment nodes. The transfer speed of the transshipment nodes varies among 0.2, 0.333 and 0.5 packets per hop. We experiment with various configurations of the 20 transshipment nodes, of which some sample configurations are shown in Fig. 9.

All TD-FALCON agents employ the bounded Q-learning (Eq. 10) and the following parameter values: learning rate $\alpha = 0.5$, discount factor $\gamma = 0.95$, and initial Q-values are set to 0. For action selection, the decaying ϵ-greedy policy is used with ϵ initialized to 0.6 and decaying linearly at a rate of 0.0004.

Figure 10 shows the success rate of 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, 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 BTF has the highest convergence rate, due to the policy sharing mechanism used by the TD-FALCON BTF algorithm. The BTF team can achieve more than 90% success rate 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

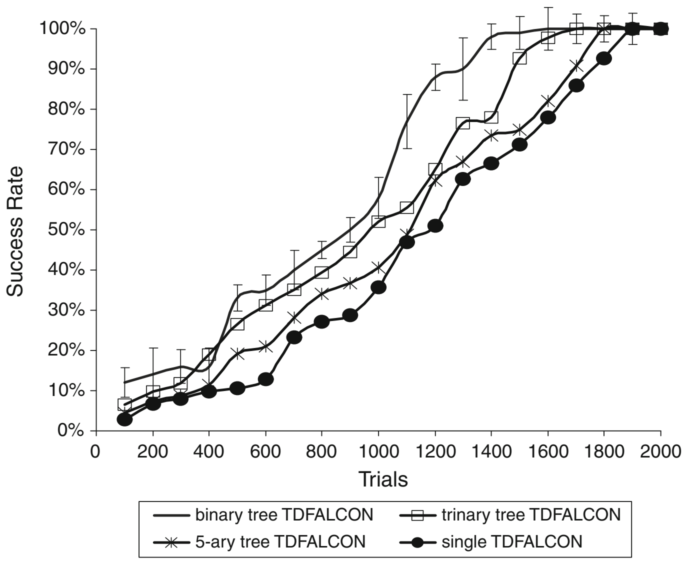

Fig. 10 Success rate of various TD-FALCON-based strategies

about 1,800 trials. Since the agents cannot share cognitive nodes and some of them may need to handle a large state space, the slow convergence of the single agent team is unavoidable.

Figure 11 shows the number 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, because any effective method of transferring objects cannot be shared across the routing nodes in the network environment in time. In contrast, a routing node equipped with a TD-FALCON binary tree group is able to transfer an object to the right successor transshipment node within a short time, due to the faster learning of the optimal path by the TD-FALCON BTF team. Therefore, 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 take, on average, about 53 hops to fulfill a task, in contrast to 54.7 hops from the single TD-FALCON team.

Figure 12 shows the number 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 n, the number of subnodes of a TD-FALCON n-ary tree. 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. The reason behind the phenomenon is that a TD-FALCON n-ary tree may face an exponentially increasing state space, as indicated in Table 4. It can also be noticed that the single TD-FALCON team generates much more cognitive nodes than the TD-FALCON tree teams, because some routing nodes face a large number of successors. 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 13 demonstrates the running time (s) in completing a routing task per trial of TD-FALCON teams in the single agent and various tree formations. We compute the average running time per trial after every 100 trials over the entire simulation of 2000 trials. The

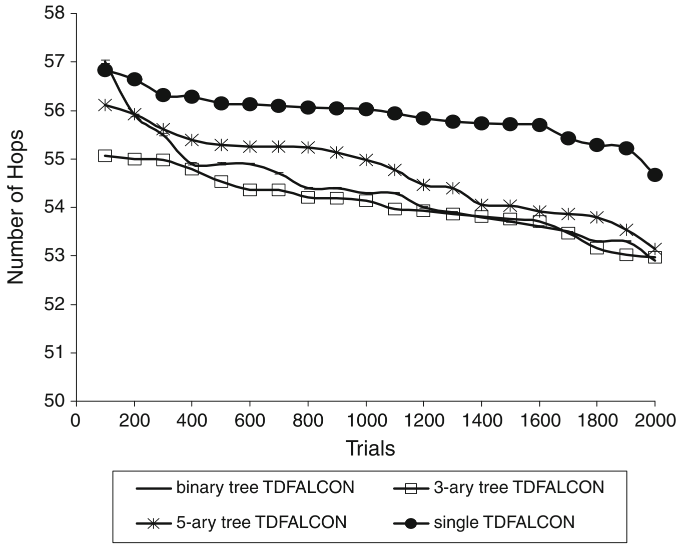

Fig. 11 Number of hops taken by various TD-FALCON-based strategies

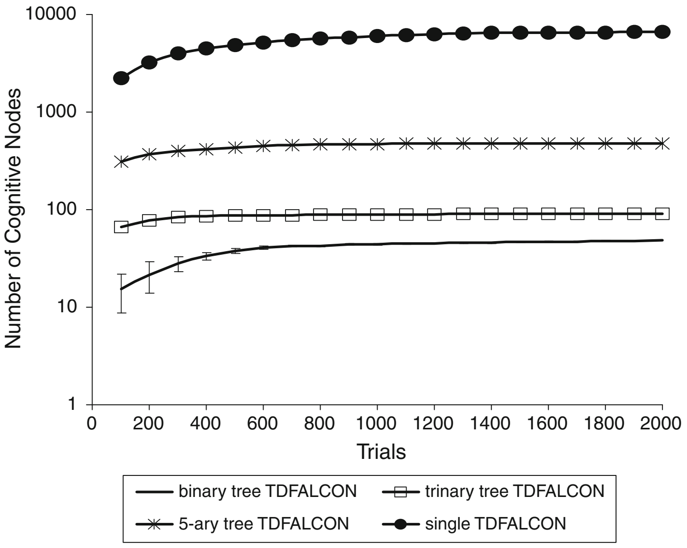

Fig. 12 Number of cognitive nodes created by various TD-FALCON based strategies

running time is based on the platform of Pentium(R) 4 CPU 2.80 GHz with 1.99 GB of RAM. We can notice an obvious trend of all tree structure teams that the running time increases with the growth of cognitive nodes, as the system has to spend more time in searching through the full set of cognitive nodes to select an action. The other noticeable trend is that a TD-FALCON n-ary tree system runs slower with the growth of n, because of the exponentially increasing state space. For example, after 2,000 trials, the TD-FALCON 5-ary tree team spends about 7.59 s in running one trial, almost twelve times (one order of magnitude) slower than the TD-FALCON BTF team, which uses only 0.612 s for one trial. It can be noticed that the

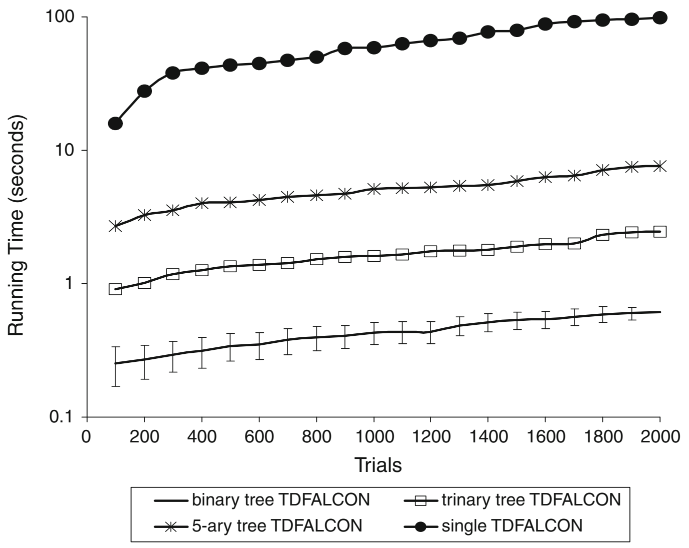

Fig. 13 Running time (s) of various TD-FALCON-based strategies

single TD-FALCON team spends tremendously more time than the TD-FALCON teams with tree formations. The reason is that the single agent team has much more cognitive nodes to go through in each cycle of performing and learning. After 2,000 trials, the single TD-FALCON team needs to take more than 98 s to run one trial, about 160 times (two orders of magnitude) slower than the TD-FALCON BTF team.

5.2 Comparing TD-FALCON with Q-learning

In this group of experiments, we compare the performance between the TD-FALCON method and the original Q-Learning method based on the table lookup mechanism [58,59]. For efficiency consideration, our implementation of Q-learning creates the entries in the table dynamically one at a time, only when a new Q-tuple is to be learned. The Q-value table thus expands dynamically during the learning process. Both TD-FALCON and Q-learning systems use the BTF, with the configuration of 50 packets and 20 transshipment nodes.

Figure 14 shows the success rate of the TD-FALCON team and the Q-learning team, averaged at 100-trial intervals across 2,000 trials, in the NR domain. It can be seen that both of the teams can eventually achieve 100% success rate. However, the TD-FALCON team can significantly outperform the Q-learning team from the very beginning and over most part of the learning period. For example, after 500 trials, the TD-FALCON team can gain 33% success rate, but the Q-learning team can only get 14% success rate, and after 1,000 trials, the TD-FALCON and the Q-learning teams can obtain 58.0% and 46.6% success rate respectively. In addition, it is also noticeable that the TD-FALCON team converges faster than the Q-learning team. After about 1,300 trials, the TD-FALCON team can gain more than 90% success rate. In contrast, the Q-learning team fails to achieve the same level of success rate until 1,500 trials. The poorer performance and the slower convergence of the Q-learning team are due to the mechanism that it has to learn a large lookup table, causing a scalability problem.

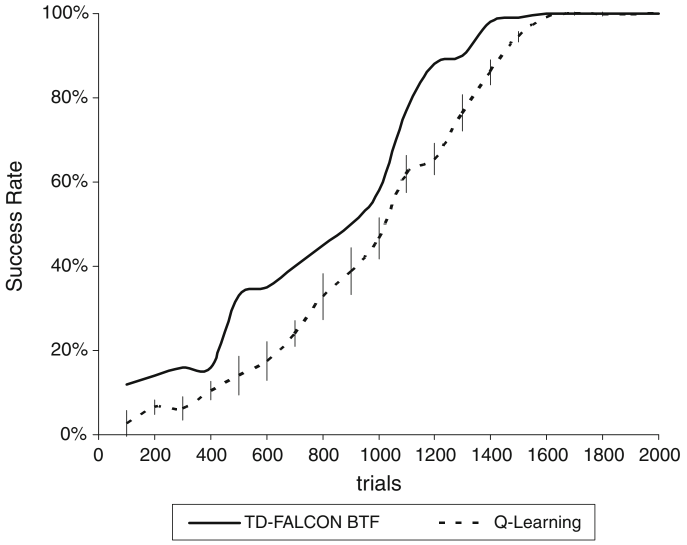

Fig. 14 Success rate of the TD-FALCON and the Q-learning teams

Figure 15 shows the number of hops taken by the TD-FALCON team and the Q-learning team, averaged at 100-trial intervals across 2,000 trials. It can be noticed that at the initial stages, the Q-learning team prominently needs more hops to fulfil a NR task. For example, after 200 trials, the Q-learning team takes 75.2 hops on average to complete a NR task, significantly outnumbering the 55.9 hops incurred by the TD-FALCON team. Subsequently, the performance figures of the two teams are approaching, for the number of hops used by the Q-learning team decreases more rapidly over time. However, even at the final stages, the Q-learning team still requires more hops than the TD-FALCON team. After 2,000 trials, the average numbers of hops used by the Q-learning and the TD-FALCON teams are 57.7 and 52.9 respectively. This trend highlights the slower convergence of the Q-learning algorithm, because it has a larger table to set up and maintain.

Figure 16 highlights the number of cognitive nodes and Q-tuples created by the TD-FALCON and the Q-learning teams respectively, averaged at 100-trial intervals across 2,000 trials on the NR task. From their performance curves, it can be noticed that both of the teams have their number of cognitive nodes or Q-tuples increased rapidly at the very beginning. This trend should be attributed to more exploration activities during the early stages of the NR task. However, after around 900 trials, the curves of both teams go flat. In addition, we can witness that as Q-learning creates one Q-tuple for each distinct state-action pair, the Q-learning team produces significantly more Q-tuples. At the end of 2,000 trials, the Q-learning team creates 376.8 Q-tuples, nearly 8 times more than the cognitive nodes created by the TD-FALCON team.

Figure 17 indicates the running time (s) per trial of the TD-FALCON and the Q-learning teams, averaged at 100-trial intervals across 2,000 trials. The running time is obtained based on the platform of Pentium(R) 4 CPU 2.80 GHz with 1.99 GB of RAM. It can be witnessed that both teams incur a longer running time, due to the increasing number of cognitive nodes or Q-tuples to go through in the performing and learning processes. However, the Q-learning team obviously requires much more time in running the trials than the TD-FALCON team, due to the long search time required in a much larger Q-value table. At the end of 2,000 trials,

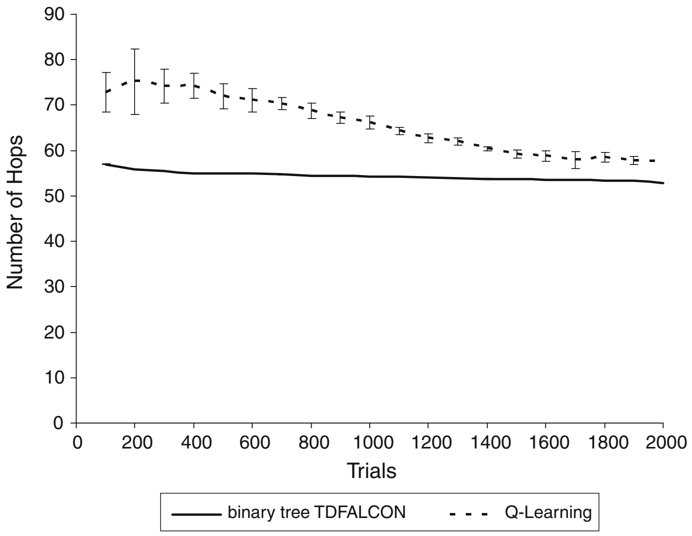

Fig. 15 Number of hops taken by the TD-FALCON and the Q-learning teams

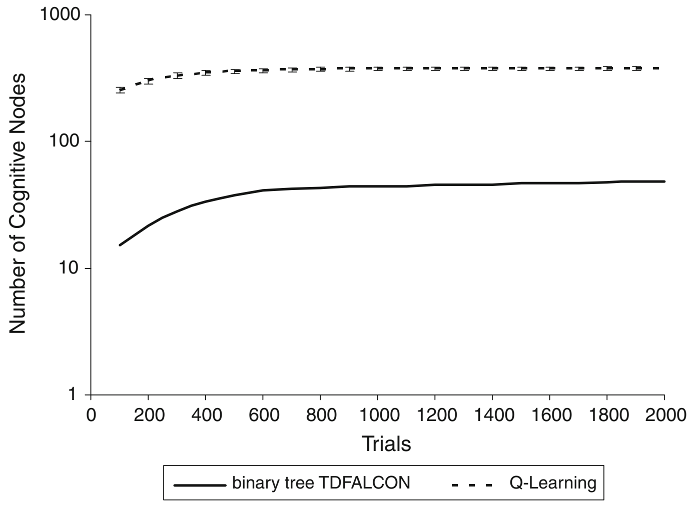

Fig. 16 Number of cognitive nodes and Q-tuples created by the TD-FALCON and the Q-learning teams respectively

the Q-learning team on average spends 92.4 s in running a trial, about 158 times (two orders of magnitude) slower than the TD-FALCON team.

5.3 Comparative experiments between TD-FALCON BTF and 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 packets and 20 transshipment nodes.

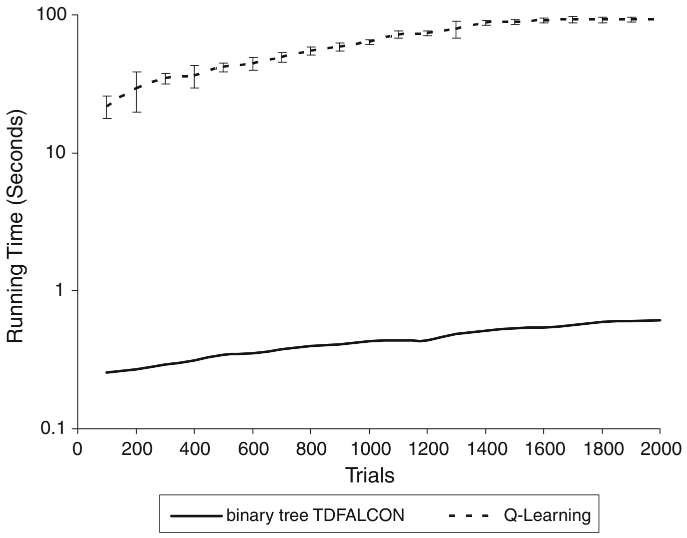

Fig. 17 Running time (s) of the TD-FALCON and the Q-learning teams

In the LP method, we install each routing node in the NR domain with a revised simplex algorithm [60,61], in which the matrix form in the formulation of Eqs. 17 and 18 is used at each iteration of the simplex algorithm.

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} \tag{17}$$

$$\text{subject to} \quad \mathbf{Ax} \le \mathbf{b},\ \mathbf{x} \ge 0 \tag{18}$$

From the perspective of a routing node, we define the NR problem as a linear program as follows. Assume that a routing node R needs to dispatch n $(n > 0)$ packets to m $(m > 0)$ successors, marked as $TS_1, TS_2, \ldots, TS_m$, and the successor TS_p $(1 \le p \le m)$ 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 $x_1, x_2, \ldots, x_m$ packets to the successor $TS_1, TS_2, \ldots, TS_m$ respectively, and there is $\sum_{p=1}^{m} x_p = n$. We define expected clearance time (ECT) of a successor TS_p $(1 \le p \le m)$ as

$$ECT_p = \frac{x_p + Q_p}{T_p} = \frac{x_p}{T_p} + \frac{Q_p}{T_p}. \tag{19}$$

Comparing with the definition of ETT in Sect. 4.1, the ECT is actually the ETT of multiple (x_p) objects in the transshipment node TS_p.

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

$$AECT = \frac{\sum_{p=1}^{m} ECT_p}{m} = \sum_{p=1}^{m} \frac{x_p}{T_p m} + \sum_{p=1}^{m} \frac{Q_p}{T_p m}. \tag{20}$$

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

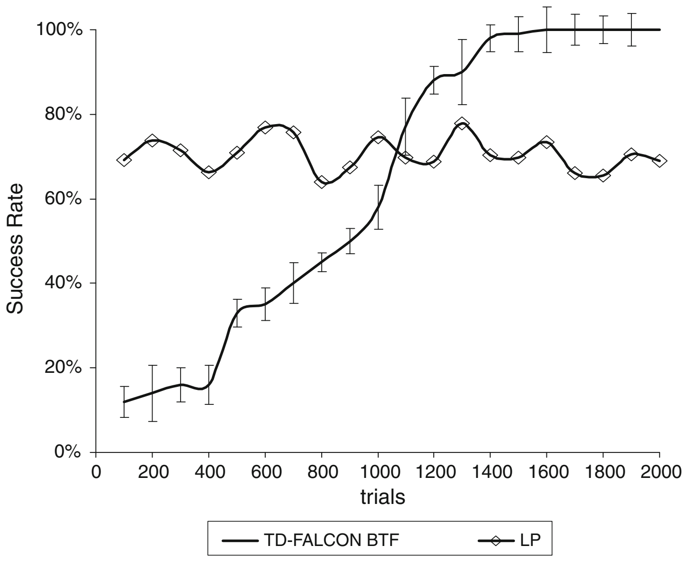

Fig. 18 Success rates of TD-FALCON BTF and LP teams

$$\text{minimize} \quad \sum_{p=1}^{m} \frac{x_p}{T_p m} + \sum_{p=1}^{m} \frac{Q_p}{T_p m} \tag{21}$$

$$\text{subject to} \quad x_p + Q_p \le C_p \, (1 \le p \le m) \tag{22}$$

$$x_p \ge 0 \, (1 \le p \le m). \tag{23}$$

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 Eqs. 17 and 18.

Note that it is hard to implement the global reward mechanism for the LP team, because each LP agent is installed locally on a routing node. As the functionality of the LP agent can only cover the successors of the routing node, it does not have the capacity to cope with the global information within its own standard input representation. If we were to use a global LP agent to collect and process all data from the entire system, this configuration would become a complex single agent problem instead of a multi-agent system.

Figure 18 shows the success rate 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 does not show further improvement. The reason is that the LP algorithm does not have an effective learning mechanism, as the TD-FALCON BTF team can provide.

Figure 19 shows the number of hops taken by 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

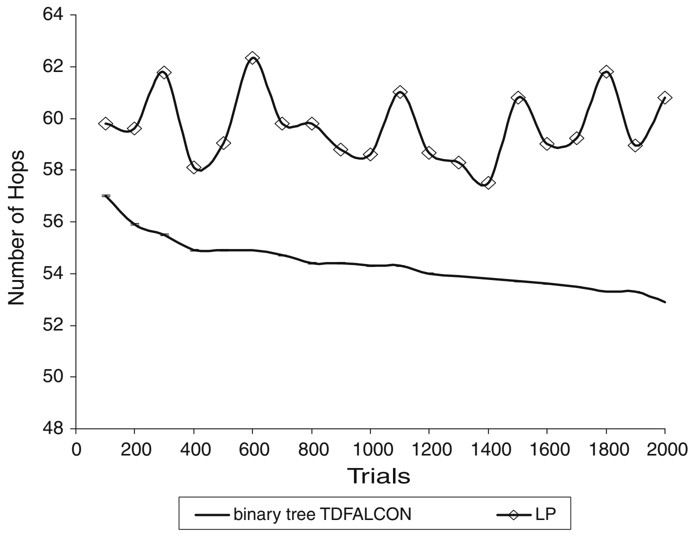

Fig. 19 Number of hops taken by the TD-FALCON BTF and the LP teams

the number of hops required by 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, but the LP team can hardly perform any learning at all. 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 20 shows the running time (s) per trial of the TD-FALCON and the LP teams, averaged at 100-trial intervals across 2,000 trials. The reported results are based on the platform of Pentium(R) 4 CPU 2.80 GHz with 1.99 GB of RAM. We can see that the TD-FALCON BTF team has nearly linear increment in running time, whereas the LP team spends significantly less time in running and its performance curve is basically flat with some fluctuations. The main reason behind the trend is that the LP team does not learn whereas TD-FALCON has to incur additional time in checking through the cognitive nodes during the learning process. At the end of 2,000 trials, the TD-FALCON BTF team on average spends 0.61188 s in running a trial, whereas the LP team spends only 0.18813 s.

5.4 Performance of TD-FALCON BTF with a varying number of packets

In this group of the experiments, we compare 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 in transferring an increasing number of packets.

In Fig. 21, we compare the success rate of a TD-FALCON BTF team in transferring 50, 80 and 100 packets, averaged at 100-trial intervals across 2,000 trials. 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, because the system may have more cases to learn with the increase in the number of packets. The TD-FALCON BTF team achieves more than 90%

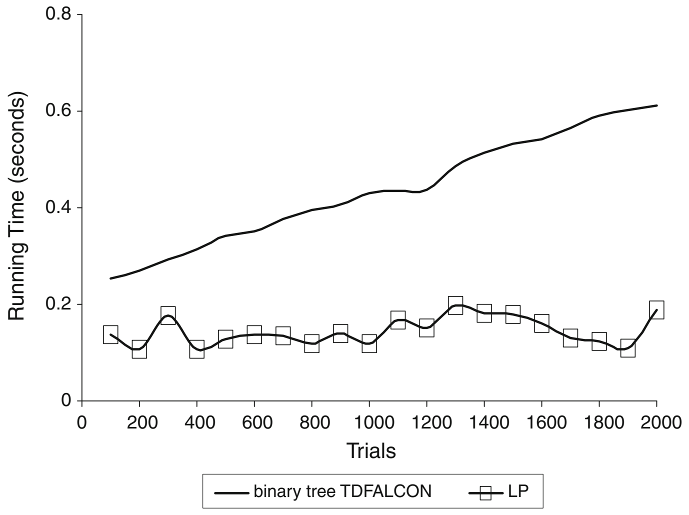

Fig. 20 Running time (s) of the TD-FALCON BTF and the LP teams

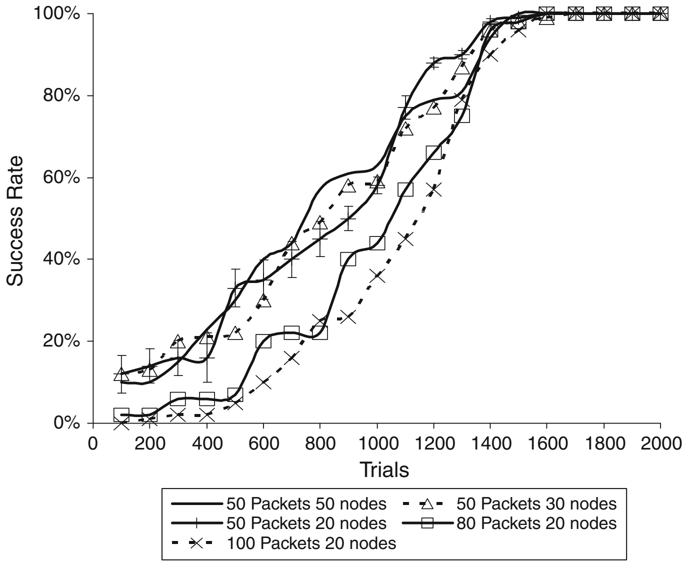

Fig. 21 Success rate of the TD-FALCON BTF team under various problem configurations

success rate within 1,300, 1,400, and 1,500 trials, for the tasks of transferring 50, 80, and 100 packets respectively.

In Fig. 22, we compare the number of hops taken by the TD-FALCON BTF team in transferring 50, 80 and 100 packets, averaged at 100-trial intervals across 2,000 trials. 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 52.9, 82.5, and 102.6 respectively. It can also be noticed that the number of hops decreases over trials, due to the effect of learning, though at a slow pace.

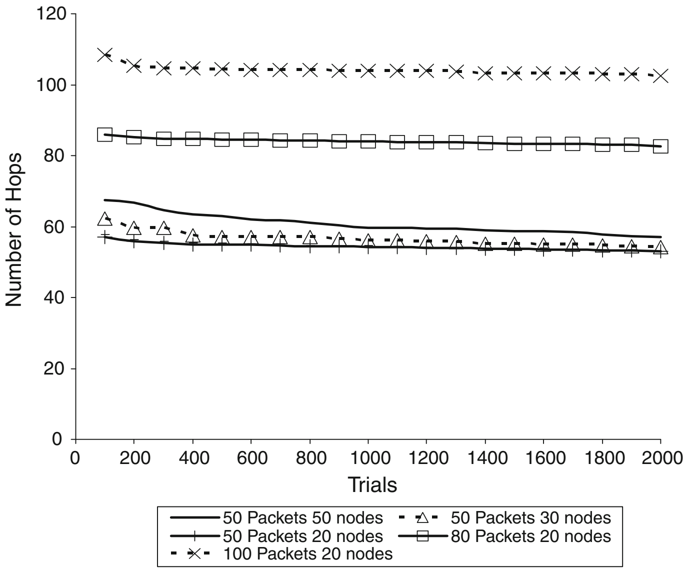

Fig. 22 Number of hops taken by the TD-FALCON BTF team under various problem configurations

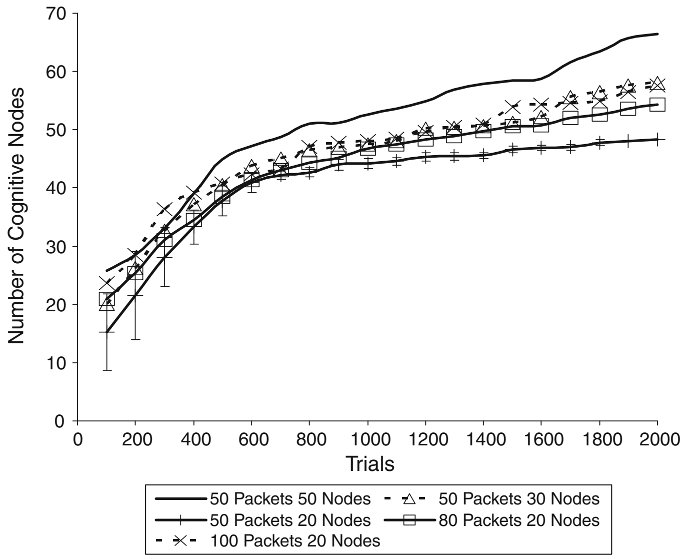

Fig. 23 Number of cognitive nodes created by the TD-FALCON BTF team under various problem configurations

In Fig. 23, we compare the number of cognitive nodes created by the TD-FALCON BTF team in transferring 50, 80 and 100 packets, averaged at 100-trial intervals across 2,000 trials. It is noticed that the number of cognitive nodes increases when more packets are transferred, due to the increasing number of cases to be learned, 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.

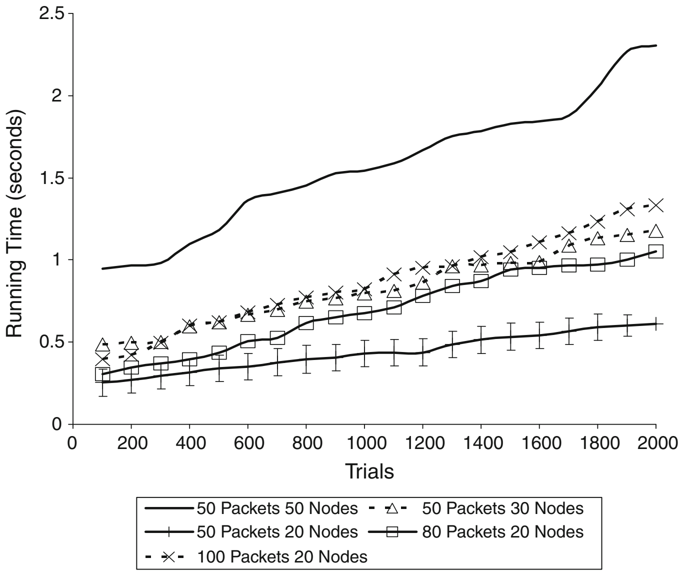

Fig. 24 Running time of the TD-FALCON BTF team under various problem configurations

In Fig. 24, we compare the running time (s) of the TD-FALCON BTF team in transferring 50, 80 and 100 packets, across 2,000 trials. It is obvious that the TD-FALCON BTF system has to run for a longer time as the number of packets to be transferred increases, due to the corresponding increase in the number of cognitive nodes. After 2,000 trials, the running times per trial of transferring 50, 80 and 100 packets are 0.612s, 1.050s and 1.332s respectively.

5.5 Performance of TD-FALCON BTF with a varying number of transshipment nodes

In this section, we compare the performance of a TD-FALCON BTF team in transferring 50 packets over NR networks composed of 20, 30 and 50 transshipment nodes. The purpose of the experiments is to test the scalability of the TD-FALCON BTF algorithm in response to an increasing number of transshipment nodes in the routing network.

Referring again to Fig. 21, we compare the success rate of the TD-FALCON BTF team in transferring 50 packets over NR networks with 20, 30 and 50 transshipment nodes respectively. It is noticed that the TD-FALCON BTF system can eventually achieve high success rate in NR networks with a varying number of transshipment nodes. This indicates that the TD-FALCON BTF algorithm can adapt to complex network environments, because of its capacity to convert multiple input sources into a uniform two-source state input. After about 1,400 trials, the TD-FALCON BTF team can achieve more than 90% success rate in various network environments.

Also in Fig. 22, we compare the number of hops taken by the TD-FALCON BTF team in transferring 50 packets over NR networks with 20, 30 and 50 transshipment nodes respectively. It is evident that the number of hops increases with the growth of the number of the transshipment nodes in the NR networks. The reason behind the trend is that there are more chances for an object to detour with the increase in the complexity of the network topology. 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, due to the learning effect.

Again in Fig. 23, we compare the number of cognitive nodes created by the TD-FALCON BTF team in transferring 50 packets over NR networks with 20, 30 and 50 transshipment nodes respectively. It is noticed that cognitive nodes increase with the growth of transshipment nodes, due to the increasing number of possible paths for an object to go, but the increment is limited. 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.

Also in Fig. 24, we compare the running time (s) of the TD-FALCON BTF team in transferring 50 packets over NR networks with 20, 30 and 50 transshipment nodes respectively. The running times of the TD-FALCON BTF team under 20, 30 and 50 transshipment nodes are 0.612s, 1.177s and 2.306s per trial respectively after 2,000 trials. It is noticed that a TD-FALCON BTF system spends less time in running in a NR network structure with fewer transshipment nodes. This is not surprising as there are fewer situations to deal with if the network structure is simpler.

6 Conclusion

In this paper, we have proposed the TD-FALCON in binary tree formation (TD-FALCON BTF) strategy by grouping a number of TD-FALCON agents in a BTF. Making use of the topological symmetry of individual nodes in a binary tree, we enable all TD-FALCON agents across the entire system to share the same set of cognitive nodes, increasing the convergence rate of all agents. The BTF can effectively convert a multi-source state inputs to a two-source problem, guaranteeing a constant number of state input vectors. Complexity analysis indicates that the TD-FALCON BTF can produce much smaller state and action spaces than other n-ary ($n > 2$) tree structures as well as the single TD-FALCON strategy. Moreover, the TD-FALCON BTF algorithm shows a high level flexibility in face of a varying number of input sources.

Comparative experiments demonstrate that the TD-FALCON BTF team can produce better performance than the Q-learning team based on the table lookup mechanism, the single TD-FALCON team and other n-ary ($n > 2$) teams, in terms of success rate, number of hops, number of cognitive nodes and running time. The experimental results show that the TD-FALCON BTF team outperforms the given approach using linear programming in terms of success rate and number of hops, but with a longer running time. From experiments, we can find that a TD-FALCON BTF team is able to keep a high level performance under various scales of NR environments.

It is interesting to note that the use of BTF is not restricted to the TD-FALCON network. Other RL methods, such as those based on gradient descent neural networks, can also adopt the knowledge sharing mechanism, as long as they have a uniform internal knowledge structure across all agents in a multi-agent system. For instance, if the multi-layer network equipped in each agent has the same number of inputs and outputs, the same number of layers as well as the same number of nodes in each layer, the networks installed are fully symmetrical, and thus the knowledge sharing mechanism can also be implemented. As a result, the BTF algorithms can be used in any multi-agent system where each agent is equipped with an identical RL network.

Nevertheless, we still prefer TD-FALCON to other RL networks, in combination with the BTF algorithm. Specifically, the learning process of a TD-FALCON agent can be accelerated significantly by using the knowledge sharing mechanism of the BTF algorithm, as the TD-FALCON model can perform online and incremental learning in a real-time environment. In contrast, in gradient descent based RL methods, the learning of new patterns may erode the previously learned knowledge. Additionally, the iterative learning which is required by the gradient descent algorithms may not be suitable in a BTF architecture because each agent may encounter very different scenarios across the episodes.

This paper has focused on the use of TD-FALCON BTF to the TMAS domains. If TD-FALCON BTF were to be used in more general multi-agent domains, it could still potentially help to improve the system performance through its policy sharing mechanism as well as the flexibility in dealing with multi-source state inputs. The main difficulty of using the TD-FALCON BTF algorithm in a more general multi-agent domain however is that the topological relationships among agents may be changing all the time. Hence we may not be able to set up a stable tree structure formation for each agent. To resolve this problem, we will have to consider the feasibility of using a dynamic TD-FALCON BTF structure for each moving agent.

In addition, our research is still restricted to homogeneous domains, where all agents are symmetrical in terms of functionalities and roles. However, there is a wider range of heterogeneous multi-agent domains. For example, a soccer game can also be regarded as a heterogeneous game, where the goal keeper and other players have different functionalities and roles. A network system can also be a heterogeneous environment, because routers, bridges and switches have quite different functionalities: whereas bridges and switches are used for transferring data packets, routers not only transfer data, but also sort the data frames and control the data flow. In a heterogeneous domain, we may still use the TD-FALCON BTF algorithm to unify and reduce the state space. However, how to apply the policy sharing mechanism across agents with different roles will be a great challenge.

References

1. Panait, L., & Luke, S. (2003). *Cooperative multi-agent learning: The state of the art*. Tech. Rep., George Mason University, Technical Report GMU-CS-TR-2003-1.
2. Busoniu, L., Babuska, R., & De Schutter, B. (2006). Multi-agent reinforcement learning: A survey. In *Proceedings of 9th international conference on control, automation, robotics and vision (ICARCV)* (pp. 1–6).
3. Lesser, V. R., Corkill, D. D., & Durfee, E. H. (1987). *An update on the distributed vehicle monitoring testbed*. Tech. Rep., Computer and Information Science Department, Amherst, MA, USA.
4. Nunes, L., & Oliveira, E. (2004). Learning from multiple sources. In *Proceedings of third international joint conference on autonomous agents and multi agent systems (AAMAS-2004).*
5. Boyan, J. A., & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 671–678). San Francisco, CA: Morgan Kaufmann Publishers Inc.
6. Chang Y. H., Ho, T., & Kaelbling L. P. (2004). Mobilized ad-hoc networks: A reinforcement learning approach. In *Proceedings of 2004 international conference on autonomic computing* (pp. 240–247).
7. Schneider, J., Wong, W. K., Moore, A., & Riedmiller, M. (1999). Distributed value functions. In *Proceedings of 16th international conference on machine learning* (pp. 371–378). San Francisco, CA: Morgan Kaufmann.
8. Varga, L. Z., Jennings, N. R., & Cockburn, D. (1994). Integrating intelligent systems into a cooperating community for electricity distribution management. *Expert Systems with Applications, 7*(4), 563–579.
9. Tan, A.-H. (2006). Self-organizing neural architecture for reinforcement learning. In *Proceedings of international symposium on neural networks (ISNN'06), LNCS 3971, Chengdu, China* (pp. 470–475).

10. Tan, A.-H., Lu, N., & Xiao, D. (2008). Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. *IEEE Transactions on Neural Networks, 9*(2), 230–244.
11. Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing, 37*, 54–115.
12. Carpenter, G. A., & Grossberg, S. (1987). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics, 26*, 4919–4930.
13. Xiao, D., & Tan, A. H. (2005). 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*.
14. Xiao, D., & Tan, A.-H. (2007). Self-organizing neural architectures and cooperative learning in multi-agent environment. *IEEE Transactions on Systems, Man, and Cybernetics-Part B, 37*(6), 1567–1580.
15. Xiao, D., & Tan, A.-H. (2008). Scaling up multi-agent reinforcement learning in complex domains. In *Proceedings of 2008 IEEE/WIC/ACM international conference on intelligent agent technology, Sydney* (pp. 326–329).
16. Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1991). Some recent advances in network flows. *SIAM Review, 33*(2), 175–219.
17. Weihmayer, R., & Velthuijsen, H. (1994). Application of distributed ai and cooperative problem solving to telecommunications. *AI Approaches to Telecommunications and Network Management*.
18. Brauer, W., & Weiß, G. (1998). Multi-machine scheduling—a multi-agent learning approach. In *Proceedings of the third international conference on multi-agent systems* (pp. 42–48).
19. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on autonomous agents (Agents97), New York*, 5–8 Feb 1997 (pp. 340–347). New York: ACM Press.
20. Riley, P., & Veloso, M. (2000). On behavior classification in adversarial environments. *Distributed Autonomous Robotic Systems, 4*, 371–380.
21. Steeb, R., Cammarata, S., Hayes-Roth, F., Thorndyke, P., & Wesson, R. (1988). Distributed intelligence for air fleet control. *Readings in Distributed Artificial Intelligence, 101*(3), 90–101.
22. Huang, J., Jennings, N. R., & Fox, J. (1995). An agent architecture for distributed medical care. *Intelligent Agents: Theories, Architectures, and Languages*, LNAI 890.
23. Chalupsky, H., Gil, Y., Knoblock, C. A., Lerman, K., Oh, J., Pynadath, D., Russ, T., & Tambe, M. (2002). Electric elves: Agent technology for supporting human organizations. *AI Magazine, 23*.
24. Crawford, E., & Veloso, M. (2004). Opportunities for learning in multi-agent meeting scheduling. In *Proceedings of artificial multiagent learning, Carnegie Mellon University, Pittsburgh, PA, USA*, Technical Report FS-04-02.
25. Wooldridge, M., Bussmann, S., & Klosterberg, M. (1996). Production sequencing as negotiation. In *Proceedings of first international conference on the practical application of intelligent agents and multi-agent technology (PAAM-96)* (pp. 709–726).
26. Vannelli, A. (1989). An interior point method for solving the global routing problem. In *Custom integrated circuits conference, 1989, Proceedings of the IEEE 1989, San Diego, CA, USA*, 15–18 May 1989 (pp. 3.4/1–3.4/4).
27. Roling, P. C., & Visser, H. G. (2008). Optimal airport surface traffic planning using mixed-integer linear programming. *International Journal of Aerospace Engineering, 2008*, 1–11.
28. Hillier, F. S. & Lieberman, G. J. (Eds.). (2001). *Introduction to operations research*. Oakland, CA: McGraw-Hill.
29. LeBlanc, L. J., Hill, J. A., Greenwell, G. W., Czesnat, A. O., & Galbreth, M. R. (2003). Optimizing nu-kote's supply chain with linear programming. *Interfaces, 34*(2), 139–146.
30. Goutis, C. (1995). A graphical method for solving a decision analysis problem. *IEEE Transactions on Systems, Man and Cybernetics, 25*, 1181–1193.
31. Das, B. C. (2010). Effect of graphical method for solving mathematical programming problem. *Daffodil International University Journal of Science and Technology, 5*(1), 29–36.
32. Szozda, N., & Świerczek, A. (2008). The success factors for supply chains of a short life cycle product. *Total Logistic Management, 1*, 163–173.
33. Zhu, K. Q., Tan, K. C., & Lee, L. H. (2000). Heuristics for vehicle routing problem with time windows. In *Proceedings of 6th international symposium on artificial intelligence and mathematics, AMAI 2000*.
34. Sun, L.-J., Hu, X.-P., Li, Y.-X., Lu, J., & Yang, D.-L. (2008). 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).

35. Lau, R. R., & Redlawsk, D. P. (2001). Advantages and disadvantages of cognitive heuristics in political decision making. *American Journal of Political Science, 45*(4), 951–971.
36. Wang, P. (1994). Heuristics and normative models of judgment under uncertainty. *International Journal of Approximate Reasoning, 14*, 221–235.
37. Rathnasabapathy, B., & Gmytrasiewicz, P. (2002) 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.
38. Schurr, N. (2007). *Toward human-multiagent teams*, Ph.D. thesis, Faculty of the Graduate School, University of Southern California, Los Angeles, CA, USA.
39. Han, J. (2006). Network-adaptive qos routing using local information. In *Proceedings of 9th Asia-Pacific network operations and management symposium, APNOMS 2006, Pusan, Korea*, 27–29 September 2006 (pp. 190–199).
40. Munetomo, M., Takai, Y., & Sato, Y. (1997). An intelligent network routing algorithm by a genetic algorithm. In *Proceedings of fourth international conference on neural information processing* (pp. 547–550).
41. Hu, X.-B., & Paolo, E. D. (2009). An efficient genetic algorithm with uniform crossover for air traffic control. *Computers and Operations Research, 36*(1), 245–259.
42. Yeh, W.-C. (2006). An efficient memetic algorithm for the multi-stage supply chain network problem. *International Journal of Advanced Manufacturing Technology, 29*(7–8), 803–813.
43. Han, C.-W., & Nobuhara, H. (2007). Advanced genetic algorithms based on adaptive partitioning method. *Journal of Advanced Computational Intelligence and Intelligent Informatics, 11*(6), 677–680.
44. Littman, M., & Boyan, J. (1993). *A distributed reinforcement learning scheme for network routing*. Tech. Rep., Robotics Institute, Pittsburgh, PA, USA, CMU-CS-93-165.
45. Baek, J.-G., Kim, C. O., & Kwon, I.-H. (2006). An adaptive inventory control model for a supply chain with nonstationary customer demands. *Computers and Operations Research, 4099/2006*, 895–900.
46. Wan, A. D. M., & Braspenning, P. J. (1995). The bifurcation of DAI and adaptivism as synthesis. In *Proceedings of the 1995 Dutch conference on AI (NAIC)* (pp. 253–262).
47. Leopold, T., Kern-Isberner, G., & Peters, G. (2008). Combining reinforcement learning and belief revision—a learning system for active vision. In *Proceedings of the 19th British machine vision conference, Leeds, UK*, 1–4 Sep 2008.
48. Stephan, V., Debes, K., Gross, H.-M., Wintrich, F., & Wintrich, H. (2000). 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* (Vol. 6, pp. 217–222).
49. Bradley, J., & Hayes, G. (2005). Adapting reinforcement learning for computer games: Using group utility functions. In *Proceedings of IEEE symposium on computational intelligence and games, Colchester, Essex, UK*.
50. Tan, A.-H., & Xiao, D. (2005). Self-organizing cognitive agents and reinforcement learning in a multi-agent environment. In *Proceedings of IEEE/ACM/WIC international conference on intelligent agent technologies* (pp. 351–357).
51. Wolpert, D. H., Tumer, K., & Frank, J. (1999). Using collective intelligence to route internet traffic. In *Proceedings of the 1998 conference on advances in neural information processing systems II, Cambridge, MA, USA* (pp. 952–958). Cambridge, MA: MIT Press.
52. Subramanian, D., Druschel, P., & Chen, J. (1997). Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceedings of fifteenth international joint conference on artificial intelligence (IJCAI-97)* (pp. 832–838). San Francisco, CA: Morgan Kaufmann.
53. Moore, B. (1988). ART 1 and pattern clustering. In *Proceedings of 1988 connectionist models summer school* (pp. 174–185).
54. Carpenter, G. A., Grossberg, S., & Rosen, D. B. (1991). Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks, 4*, 759–771.
55. Tan, A.-H. (2007). Direct code access in self-organizing neural architectures for reinforcement learning. In *Proceedings, international joint conference on artificial intelligence (IJCAI07), Hyderabad, India* (pp. 1071–1076).
56. Pérez-Uribe, A. (2002). *Structure-adaptable digital neural networks*, Ph.D. thesis, Swiss Federal Institute of Technology-Lausanne.
57. Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of 10th international conference on machine learning* (pp. 330–337).
58. Onat, A. (1998). Q-learning with recurrent neural networks as a controller for the inverted pendulum problem. In *Proceedings of the fifth international conference on neural information processing, Japan*, 21–23 Oct 1998 (pp. 837–840).

59. Sandholm, T., & Crites, R. H. (1995). On multiagent q-learning in a semi-competitive domain. In *Proceedings of the workshop on adaption and learning in multi-agent systems (IJCAI'95), London, UK* (pp. 191–205). London: Springer.
60. Benhamadou, M. (2002). On the simplex algorithm 'revised form. *Advances in Engineering Software, 33*(11–12), 769–777.
61. Dantzig, G. B., Orden, A., & Wolfe, P. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics, 5*(2), 183–195.