Self-organizing neural networks integrating domain knowledge and reinforcement learning

Teck-Hou TENG
Nanyang Technological University

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

Jacek M. ZURADA
University of Louisville

Self-Organizing Neural Networks Integrating Domain Knowledge and Reinforcement Learning

Teck-Hou Teng, *Member, IEEE*, Ah-Hwee Tan, *Senior Member, IEEE*, and Jacek M. Zurada, *Life Fellow, IEEE*

***Abstract*— The use of domain knowledge in learning systems is expected to improve learning efficiency and reduce model complexity. However, due to the incompatibility with knowledge structure of the learning systems and real-time exploratory nature of reinforcement learning (RL), domain knowledge cannot be inserted directly. In this paper, we show how self-organizing neural networks designed for online and incremental adaptation can integrate domain knowledge and RL. Specifically, symbol-based domain knowledge is translated into numeric patterns before inserting into the self-organizing neural networks. To ensure effective use of domain knowledge, we present an analysis of how the inserted knowledge is used by the self-organizing neural networks during RL. To this end, we propose a vigilance adaptation and greedy exploitation strategy to maximize exploitation of the inserted domain knowledge while retaining the plasticity of learning and using new knowledge. Our experimental results based on the pursuit-evasion and minefield navigation problem domains show that such self-organizing neural network can make effective use of domain knowledge to improve learning efficiency and reduce model complexity.**

***Index Terms*— Adaptive resonance theory (ART), domain knowledge, reinforcement learning (RL), self-organizing neural networks.**

I. Introduction

THE use of domain knowledge in machine learning has become pervasive over the past two decades largely due to its significant impact on learning outcomes [1]. Integrating domain knowledge that cannot be easily learned into a learning model can lead to improvements in terms of both learning efficiency as well as model complexity [2]. When designing real-time autonomous systems, knowledge insertion is especially beneficial as it allows the systems to start operating at a reasonable level of performance in a potentially hostile environment.

Various approaches to incorporating domain knowledge into different learning processes have been proposed. Early works on the use of domain knowledge include training neural networks by learning from hints [3], the first-order combined learner in inductive learning [4], the explanation-based neural network in explanation-based learning [5], the knowledge-based artificial neural network (KBANN) [6], and the knowledge-based conceptual neural network (KBCNN) [7] in supervised learning. Although many different usages of domain knowledge have been demonstrated, few of them attempted to insert symbolic knowledge explicitly into the learning models. This is thought to be largely due to the difficulty in reconciling the human-specified symbolic knowledge and the specialized knowledge representation typically found in the learning models. For reinforcement learning (RL) systems, there is also a dilemma between exploiting inserted knowledge and exploring new knowledge.

In most RL systems, domain knowledge is used indirectly to improve learning efficiency [8]–[11]. In contrast, this paper shows how domain knowledge can be directly inserted into a self-organizing neural network model known as fusion architecture for learning and cognition (FALCON) [12], [13] and used for RL. As a generalized form of adaptive resonance theory (ART) [14], FALCON performs online incremental learning of cognitive nodes for encoding value and action policies based on evaluative feedback from the environment. As the knowledge encoded by the cognitive nodes in FALCON is compatible with symbolic rule-based representation, IF-THEN rules specified by human experts can be inserted into FALCON, and subsequently, refined and supplemented by the discovered knowledge.

While our initial experiments on rule insertion have yielded encouraging results, they are limited to relatively simple domains, such as minefield navigation task (MNT) [15] and route planning task [16]. Issues arise when the method was applied to problems involving larger state spaces. Specifically, rather than making use of the inserted knowledge, which are highly generalized patterns of the state space, the model tends to create and use newly learned knowledge.

To improve the use of the inserted knowledge, we present an analysis to show how the vigilance parameters of FALCON can influence the exploitation of inserted knowledge and the learning of new knowledge patterns. From the analysis, we conclude that using low state vigilance encourages exploitation of domain knowledge. More importantly, we propose a reward vigilance adaptation strategy that enables the maximal exploitation of domain knowledge while retaining the flexibility of exploring new knowledge if necessary. This strategy depends on the reward vigilance criterion to ensure the selection of good rules. Consequentially, exploration of new knowledge is conducted only when no cognitive nodes can satisfy the reward vigilance criterion.

We evaluate our proposed strategies using a nontrivial pursuit-evasion (PE) problem domain and the MNT. The first

Manuscript received July 17, 2013; revised May 13, 2014; accepted May 24, 2014.

T.-H. Teng and A.-H. Tan are with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: thteng@ntu.edu.sg; asahtan@ntu.edu.sg).

J. M. Zurada is with the University of Louisville, Louisville, KY 40292 USA, and also with the Information Technology Institute, University of Social Sciences, Łódź 90-113, Poland (e-mail: jacek.zurada@louisville.edu).

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

Digital Object Identifier 10.1109/TNNLS.2014.2327636

set of experiments investigates the impact of using different state vigilances on the inserted domain knowledge. The next set of experiments investigates the effect of using greedy exploitation with the reward vigilance adaptation strategy. The third set of experiments compares the greedy exploitation and adaptive reward vigilance strategy with alternative approaches, including a naïve response system, a nonadaptive FALCON, an adapted k nearest neighbor (NN)-TD(λ) model [17], an adapted growing self-organizing map (GSOM) [18], and standard Q-learning. The final experiment evaluates the proposed strategies in the MNT problem domain. The experimental results confirm our hypothesis that a self-organizing neural network that makes proper use of domain knowledge can have improved learning efficiency and reduced model complexity.

The presentation of this approach opens with a survey of related works that use domain knowledge to improve learning efficiency in Section II. This is followed by a summary of FALCON in Section III. Details on how domain knowledge can be used to improve learning efficiency are provided in Section IV. The PE problem domain is introduced in Section V. This is followed by the presentation of the experiments and the results in Section VI. The conclusion of this paper is provided in Section VII.

II. Related Work

Various approaches to integrate domain knowledge and different types of learning systems have been known. In many of these works, domain knowledge is used in an indirect manner. For example, domain knowledge has been used to generate rare examples for learning [19]. Domain knowledge were also used for selecting variables and features for learning systems [20]. Direct use of different types of domain knowledge in inductive learning was also known to significantly reduce the amount of search required [4].

Bayesian networks, on the other hand, have used domain knowledge in both direct and indirect ways. Direct use of domain knowledge in Bayesian networks includes [21], where domain knowledge expressed intuitively using object-oriented Bayesian network was used with the structural EM algorithm for improving learning efficiency. Indirect uses of domain knowledge in Bayesian networks includes [22], where domain knowledge in the form of Bayesian prior probability distribution within the preferential metric was used to derive the instantaneous cost of incremental learning using gradient-descent algorithms. In [23], domain knowledge from experts was used in search-based learning process of Bayesian networks.

Indirect uses of domain knowledge in neural networks includes [3], where knowledge on known properties of a function was used in learning-from-examples paradigm for training neural networks. In [24], transformation-invariance domain knowledge and knowledge on data were incorporated into support vector machines to build invariant kernels, generate training data, and formulate problem of optimization methods. In addition, domain knowledge was used to design a new class of neural networks that generalized better than standard artificial neural network [25].

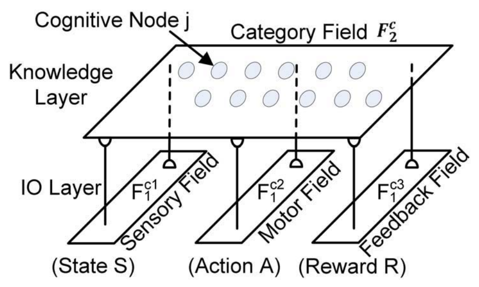


Fig. 1. FALCON architecture.

Direct uses of domain knowledge in neural networks for supervised learning includes [7], where domain knowledge is inserted into KBCNN and refined using supervised learning. In [6], symbolic rules were used to initialize the network structure of KBANN. However, refinement of the learned symbolic rules using backpropagation algorithm can erode the originally inserted knowledge. To address this problem, cascade ARTMAP was proposed to integrate domain knowledge represented as propositional rules and neural network learning [2]. Based on ART, cascade ARTMAP enabled stable encoding of inserted knowledge and further learning of new knowledge when necessary by creating new cognitive nodes.

Domain knowledge is typically used indirectly in RL systems. For example, domain knowledge on the rate of spatial variation of the action values was incorporated into Q-learning [8]. In addition, domain knowledge was used to initialize the evaluation function and to define state-dependent action sets [9]. In addition, domain knowledge was used to partition the key states to speed up RL [10]. An example of the direct use of domain knowledge is when domain knowledge in the form of approximate plans with several choices points was incorporated into Icarus agent for further refinement using RL [11].

As seen, many learning systems have used domain knowledge in an indirect fashion. There are even fewer attempts to insert domain knowledge directly into real-time learning models. This, we believe, is due to the difficulty in reconciling domain knowledge and real-time learning. For RL, there is an added dilemma of balancing between exploitation of inserted knowledge and the exploration of new knowledge.

III. Reinforcement Learning Model

FALCON [12], [13] is a class of self-organizing neural network designed for RL. Based on ART [14], FALCON learns incrementally and generalizes in real time. The long-term value of the selected action choices to the states is estimated using a temporal difference (TD) method known as bounded Q-learning [26].

A. Structure and Operating Modes

Structurally, the FALCON neural network employs a three-channel architecture (Fig. 1) comprising an input/output (IO) layer and a knowledge layer. The IO layer has three input fields, namely a sensory field F_1^{c1} for accepting state vector $\mathbf{S}$,

Algorithm 1 FALCON Algorithm

```
Require: Activity vectors $\mathbf{x}^{ck}$ and all weight vectors $\mathbf{w}_j^{ck}$ for $k = \{1, 2, 3\}$
1: for each $F_2^c$ node $j$ do
2:     Code Activation: The choice function $T_j^c$ is derived using
           $$T_j^c = \sum_{k=1}^{3} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|} \quad (1)$$
       where the fuzzy AND operation $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, the norm $|\cdot|$ is
       defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$, $\alpha^{ck} \in [0, 1]$ are the choice
       parameters, and $\gamma^{ck} \in [0, 1]$ are the contribution parameters.
3: end for
4: repeat
5:     Code Competition: Index of winning cognitive node $J$ is found using
           $$J = \arg\max_j \{T_j^c : \text{for all } F_2^c \text{ node } j\}.$$
6:     Template Matching: Derive $m_J^{ck}$ to determine whether resonance is
       attained using
           $$m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck} \quad (2)$$
       where $\rho^{ck} \in [0, 1]$ are the vigilance parameters.
7:     if vigilance criterion is satisfied then
8:         Resonance is attained.
9:     else
10:        Match Tracking: Modify state vigilance $\rho^{c1}$ using
               $$\rho^{c1} = \min\{m_J^{ck} + \psi, 1.0\}$$
           where $\psi$ is a very small step increment to match function $m_J^{ck}$.
11:        Reset: $T_J^c = 0.0$.
12:    end if
13: until Resonance State is attained.
14: if operating in LEARN/INSERT mode then
15:    Template Learning: Modify $\mathbf{w}_J^{ck}$ using
           $$\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})}) \quad (3)$$
       where $\beta^{ck} \in [0, 1]$ are the learning rate parameters.
16: else if operating in PERFORM mode then
17:    Activity Readout: Read out the action vector $\mathbf{A}$ of cognitive node $J$
       using
           $$\mathbf{x}^{c2(\text{new})} = \mathbf{x}^{c2(\text{old})} \wedge \mathbf{w}_J^{c2}. \quad (4)$$
       Decode $\mathbf{x}^{c2(\text{new})}$ to derive action choice $a$.
18: end if
```

an action field F_1^{c2} for accepting action vector $\mathbf{A}$, and a reward field F_1^{c3} for accepting reward vector $\mathbf{R}$. The category field F_2^c at the knowledge layer stores the committed and uncommitted cognitive nodes. Each cognitive node j has a set of template weights $\mathbf{w}_j^{ck}$ for $k = \{1, 2, 3\}$.

FALCON operates in the PERFORM mode to decide on action choices for the states. FALCON operates in the LEARN mode to learn the long-term values of these action choices on the states. To be detailed in Section IV-A, FALCON operates in the INSERT mode to assimilate domain knowledge.

Given an input state, presented as activity vector $\mathbf{x}^{ck}$, FALCON selects winning cognitive node J using the approach outlined in Algorithm 1. In the PERFORM mode, the action field is used in activity readout to decode $\mathbf{w}_J^{c3}$ as action choice a using (4). Selecting an uncommitted cognitive node in the PERFORM mode triggers exploration for action choice a. In the LEARN mode, cognitive node J is used in template learning to learn the activity vectors $\mathbf{x}^{ck}$ using (3). A new cognitive node is created when an uncommitted cognitive node is selected as the winning node in the LEARN mode.

Algorithm 2 TD-FALCON Algorithm

```
1: Initialize the FALCON network.
2: Sense the environment and formulate a state vector $\mathbf{S}$ based on the current
   state $s$.
3: Following an action selection policy, first make a choice between explo-
   ration and exploitation.
4: if Exploration then
5:     Choose action choice $a$ using an exploration strategy.
6: else if Exploitation then
7:     Identify action choice $a$ with the maximal $Q(s, a)$ value by presenting
       the state vector $\mathbf{S}$, the action vector $\mathbf{A} = \{1, \ldots, 1\}$, and the reward
       vector $\mathbf{R} = \{1, 0\}$ to FALCON.
8: end if
9: Perform the action choice $a$, observe the next state $s'$, and receive a reward
   $r$ (if any) from the environment.
10: Estimate the revised value function $Q(s, a)$ following a TD formula, such
    as $\Delta Q(s, a) = \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$.
11: Formulate action vector $\mathbf{A}$ based on action choice $a$ and reward vector $\mathbf{R}$
    based on $Q(s, a)$.
12: Present the corresponding state $\mathbf{S}$, action $\mathbf{A}$, and reward $\mathbf{R}$ vectors to
    FALCON for learning.
13: Update the current state by $s = s'$.
14: Repeat from Step 2 until $s$ is a terminal state.
```

B. Incorporating TD Method

For learning from delayed evaluative feedback signals, the value function $Q(s, a)$ of state-action pairs is estimated using TD method outlined in Algorithm 2 [13]. At time t, lines 1–9 of Algorithm 2 show FALCON operating in PERFORM mode to select action choice a either by exploration or by exploitation. At time $t + 1$, lines 10–13 of Algorithm 2 show that FALCON operating in LEARN mode uses reward r from the environment on action choice a to estimate the value function $Q(s, a)$.

1) Iterative Value Estimation: A TD method known as bounded Q-learning [13] is iteratively used to estimate the value of applying action choice a to state s. The Q-value update function is given by

$$Q^{\text{new}}(s, a) = Q(s, a) + \alpha \text{TD}_{\text{err}}(1 - Q(s, a))$$

where $\alpha \in [0, 1]$ is the learning parameter and the TD_{err} is the temporal error term, which is derived using

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

where $\gamma \in [0, 1]$ is the discount parameter and the $\max_{a'} Q(s', a')$ is the maximum estimated value of the next state s'. The estimated Q-value $Q^{\text{new}}(s, a)$ is used as a teaching signal to learn the association of state s and action choice a. It is notable that in TD-FALCON, the values of $Q(s, a)$ and $\max_{a'} Q(s', a')$ are in turn estimated using the same FALCON network.

C. Action Selection Policy

The action selection policy used here is known as self-regulating action exploration [27]. Using this approach, action exploration is performed with a probability of ϵ, where $\epsilon \in [0, 1]$ is regulated by the performance of the learning system. The performance is estimated using an interval success rate

ϕ derived using $\phi = w_s/w_n$, where w_s is the number of successful trials within w_n training iterations. After every w_n trials, ϵ is updated using $\epsilon = 1 - \phi$.

In addition, after each update, ϵ is linearly decayed over the next w_n training iterations using an ϵ-decay rate δ, which is derived using ϵ/w_n. As exploitation is performed with a probability of $1 - \epsilon$, such an approach gradually increases exploitation of the learned knowledge within w_n training iterations.

D. Pruning

RL can lead to the learning of cognitive nodes that become irrelevant after some time. Action selection and learning become inefficient when these irrelevant cognitive nodes are not pruned. Therefore, a confidence-based pruning strategy similar to the one proposed in [12] is adopted to prune the irrelevant cognitive nodes.

Specifically, each cognitive node j has a confidence level c_j where $c_j \in [0.0, 1.0]$ and an age σ_j where $\sigma_j \in [0, \mathcal{R}]$. A newly committed cognitive node j has an initial confidence level $c_j(0)$ and an initial age $\sigma_j(0)$. At time t, the confidence level c_J of winning cognitive node J is reinforced using

$$c_J(t+1) = c_J(t) + \eta(1 - c_J(t))$$

where η is the reinforcement rate of the confidence level for all cognitive nodes. Also at time t, the confidence level c_j of cognitive node j is decayed using

$$c_j(t+1) = c_j(t) - \zeta c_j(t)$$

where ζ is the decay rate of the confidence level for all cognitive nodes.

An age attribute σ_j is introduced to cognitive node j to prevent it from being pruned when $\sigma_j = \sigma_j(0)$, $c_j = c_j(0)$ and $c_j < c^{\text{rec}}$, where c^{rec} is the recommended confidence threshold. At time t, age σ_j of cognitive node j is modified using $\sigma_j(t)+1$. Cognitive node j is pruned only when $c_j(t) < c^{\text{rec}}$ and $\sigma_j(t) \geq \sigma^{\text{old}}$, where σ^{old} is the old age threshold.

IV. Bootstrap Reinforcement Learning Using Domain Knowledge

A. Inserting Domain Knowledge

Domain knowledge defined using symbols has to be translated into vector patterns before insertion into FALCON. The presentation of this two-step process of inserting domain knowledge begins with the following definition of propositional rules.

Definition 1 (Propositional Rule): Given a state space

$$\mathbf{X} = \{x_1, x_2, \ldots, x_n, \ldots, x_N\}$$

and an action space

$$\mathbf{Y} = \{y_1, y_2, \ldots, y_m, \ldots, y_M\}$$

where $\mathbf{X} \cap \mathbf{Y} \equiv 0$, and each unit of domain knowledge is defined as a propositional rule r_j with the following format:

$$\text{Rule } r_j : \text{IF } \mathbf{X}^{r_j} \text{ THEN } \mathbf{Y}^{r_j} (\text{REWARD } p^{r_j})$$

Algorithm 3 Translation of Propositional Rules

```
Ensure: Initialize FALCON with an uncommitted cognitive node.
 1: for each propositional rule $r_j$ do
 2:   for each attribute $a_p \in \mathbf{X}^{r_j}$ do
 3:     for each attribute-value binding $b_{pq} \in V(a_p)$ do
 4:       Translate $b_{pq}$ into vector $\mathbf{v}_{pq}$ using (5).
 5:     end for
 6:     Translate $a_p$ into attribute vector $\mathbf{S}_p$ using (6).
 7:   end for
 8:   Translate antecedent $\mathbf{X}^{r_j}$ into state vector $\mathbf{S}^{r_j}$ using (7).
 9:   Repeat steps 3–7 for translation of each attribute $a_p \in \mathbf{Y}^{r_j}$.
10:   Translate consequent $\mathbf{Y}^{r_j}$ into action vector $\mathbf{A}^{r_j}$ using (8).
11:   Set reward $p^{r_j}$ into reward vector $\mathbf{R}^{r_j}$ using (9).
12:   Operate FALCON in INSERT mode to insert translated propositional
      rule $r_j$ as $\{\mathbf{S}^{r_j}, \mathbf{A}^{r_j}, \mathbf{R}^{r_j}\}$.
13: end for
14: return FALCON with inserted domain knowledge.
```

where the antecedent set $\mathbf{X}^{r_j} \subset \mathbf{X}$, the consequent set $\mathbf{Y}^{r_j} \subset \mathbf{Y}$, and $p^{r_j} \in [0, 1]$ is the reward factor.

From Definition 1, let a_p be an attribute in either state space $\mathbf{X}$ or action space $\mathbf{Y}$, the set of possible attribute-value binding b_{pq} of attribute a_p is defined as

$$V(a_p) = \{b_{p1}, b_{p2}, \ldots, b_{pq}, \ldots, b_{pQ}\}.$$

For $\|\mathbf{X}^{r_j}\| \geq 2$, conjunctive relationship exists among the antecedents of rule r_j, while disjunctive relationship exists between rules r_1 and r_2 with identical consequent, i.e., $\mathbf{X}^{r_1} \neq \mathbf{X}^{r_2}$ and $\mathbf{Y}^{r_1} \equiv \mathbf{Y}^{r_2}$.

1) Translation: Outlined in Algorithm 3, each propositional rule r_j has to be translated from its symbol-based representation into vector pattern for insertion into FALCON. Each attribute-value binding b_{pq} of attribute a_p in propositional rule r_j is converted into a complement-coded vector $\mathbf{v}_{pq} = \{v_{pq}, v_{pq}^c\}$, where $v_{pq}^c = 1 - v_{pq}$. Specifically, $\mathbf{v}_{pq}$ is determined as

$$\{v_{pq}, v_{pq}^c\} = \begin{cases} \{1, 0\} & \text{if } a_p \equiv b_{pq}, \\ \{0, 1\} & \text{if } a_p \neq b_{pq}, \\ \{0, 0\} & \text{if } a_p \text{ is not considered.} \end{cases} \quad (5)$$

Consequentially, each attribute a_p can be translated into a concatenated vector

$$\mathbf{S}_p = \{\mathbf{v}_{p1}, \mathbf{v}_{p2}, \ldots, \mathbf{v}_{pq}, \ldots, \mathbf{v}_{pQ}\}. \quad (6)$$

Hence, the antecedent $\mathbf{X}^{r_j}$ can be translated into a fixed-length state vector

$$\mathbf{S}^{r_j} = \{\mathbf{S}_n | n \in [1, \|\mathbf{X}\|]\} \quad (7)$$

and the consequent $\mathbf{Y}^{r_j}$ can be translated into a fixed-length action vector

$$\mathbf{A}^{r_j} = \{\mathbf{S}_m | m \in [1, \|\mathbf{Y}\|]\} \quad (8)$$

and the reward p^{r_j}, where $p^{r_j} \in \{0, 1\}$ can be translated into a two-element reward vector

$$\mathbf{R}^{r_j} = \{p^{r_j}, 1 - p^{r_j}\}. \quad (9)$$

In effect, the antecedent $\mathbf{X}^{r_j}$, consequent $\mathbf{Y}^{r_j}$, and reward p^{r_j} of the propositional rule r_j are translated into the state vector $\mathbf{S}^{r_j}$, the action vector $\mathbf{A}^{r_j}$, and the reward vector $\mathbf{R}^{r_j}$, respectively.

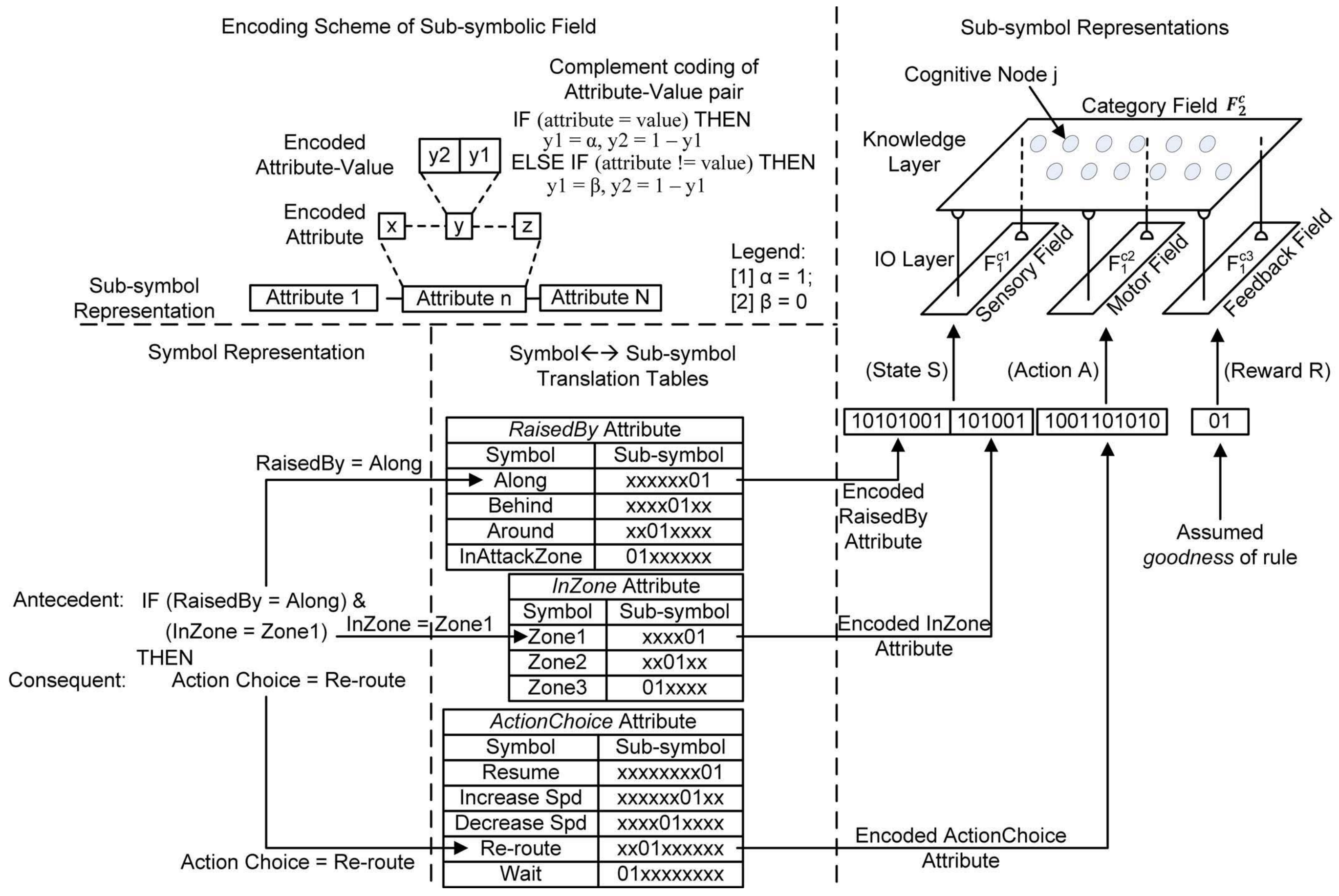


Fig. 2. Encoding of domain knowledge and inserting it into FALCON.

2) Insertion: The tuple $\{\mathbf{S}^{r_j}, \mathbf{A}^{r_j}, \mathbf{R}^{r_j}\}$ of translated propositional rule r_j is presented to the respective fields of FALCON as the activity vector $\mathbf{x}^{ck}$. To learn $\{\mathbf{S}^{r_j}, \mathbf{A}^{r_j}, \mathbf{R}^{r_j}\}$, a winning cognitive node J is selected using vigilance parameters $\rho^{ck} = \{1.0, 1.0, 1.0\}$. This means the respective weight vectors of the winning cognitive node J have to be perfectly matched to $\{\mathbf{S}^{r_j}, \mathbf{A}^{r_j}, \mathbf{R}^{r_j}\}$. An illustration of the rule translation and insertion procedures can be observed in Fig. 2.

During insertion, duplicates are eliminated by grouping translated propositional rules with identical vector patterns into the same cognitive node. In the absence of a perfectly matched committed cognitive nodes, an uncommitted cognitive node is picked to learn $\{\mathbf{S}^{r_j}, \mathbf{A}^{r_j}, \mathbf{R}^{r_j}\}$. It becomes a newly committed cognitive node, and a new cognitive node is created.

FALCON operates in the INSERT mode to assimilate the translated propositional rules. The insertion of these translated propositional rules into FALCON as cognitive nodes bootstraps FALCON for action selection and learning. When FALCON operates in the PERFORM mode, the action choice to the states is derived from a cognitive node best matched to the states. When FALCON operates in the LEARN mode, the cognitive node best matched to the previous state is picked for updating.

3) Broad Propositional Rules: Domain knowledge is often specified using selected attributes. Irrelevant attributes are excluded from the antecedent leading to a broadening of the scope of application of such propositional rules to more states. Such propositional rules are formally defined below as broad propositional rules.

Definition 2 (Broad Propositional Rule): Given that state space $\mathbf{X}$ is a set of attributes x_n where $n \in [1, \|\mathbf{X}\|]$ and the action space $\mathbf{Y}$ is a set of attributes y_m where $m \in [1, \|\mathbf{Y}\|]$, a broad propositional rule r_j has antecedent set $\mathbf{X}^{r_j}$ where $\mathbf{X}^{r_j} \subset \mathbf{X}$ and consequent set $\mathbf{Y}^{r_j}$ where $\mathbf{Y}^{r_j} \subset \mathbf{Y}$.

The antecedent set $\mathbf{X}^{r_j}$ of a broad propositional rule r_j is translated into fixed-length state vector $\mathbf{S}^{r_j}$ using the translation technique outlined in Algorithm 3. The length of state vector $\mathbf{S}^{r_j}$ is fixed because it encompasses the entire state space $\mathbf{X}$. Attribute x_n missing from antecedent $\mathbf{X}^{r_j}$ of a broad propositional rule r_j is translated into a zero attribute vector $\mathbf{S}_n^{r_j} = \{0, \ldots, 0\}$. State vector $\mathbf{S}^{r_j}$ with zero attribute vector $\mathbf{S}_n^{r_j}$ leads to a generalized state vector as defined below.

Definition 3 (Generalized State Vector): The translated antecedent $\mathbf{X}^{r_j}$ of a broad propositional rule r_j is a fixed length generalized state vector $\mathbf{S}^{r_j}$ such that $\mathbf{S}^{r_j} = \{\mathbf{S}_1^{r_j}, \ldots, \mathbf{S}_{\|\mathbf{X}\|}^{r_j}\}$, where $\mathbf{X}^{r_j} \subset \mathbf{X}$ and $\exists i \in [1, \|\mathbf{X}\|]$, $\mathbf{S_i}^{r_j} = \{0, \ldots, 0\}$.

The consequent set $\mathbf{Y}^{r_j}$ is assumed to be equivalent to the action space $\mathbf{Y}$, while the reward p^{r_j} is translated into a complement-coded two-element vector $\mathbf{R}^{r_j}$. Therefore, the concept of being broad is limited to the definition of the antecedent of the propositional rules. This is highlighted because specific design principle of the self-organizing neural network has to be followed to improve the learning efficiency from the use of domain knowledge.

B. Analysis of FALCON Using Domain Knowledge

In this section, an analysis of the characteristic of the code selection process is presented to show how FALCON balances between exploitation of inserted domain knowledge and exploration for new knowledge.

1) Code Selection: The inserted domain knowledge and the learned knowledge are subjected to the same code selection procedures for action selection and learning. Given an activity vector $\mathbf{x}^{ck}$, the choice function T_j^c of each cognitive node j is derived using (1). The property regarding the choice probability of cognitive nodes with respect to activity vector $\mathbf{x}^{ck}$ is specified in Lemma 1.

Lemma 1 (Choice Probability of Cognitive Nodes): Assuming all things equal, given two cognitive nodes j_1 and j_2 such that

$$|\mathbf{w}_{j_1}^{c1}| < |\mathbf{w}_{j_2}^{c1}|$$

the probability of cognitive node j_1 winning the code competition is higher than that of cognitive node j_2.

Proof: Following the direct code access method [28], we have $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \{1, \ldots, 1\}$, and $\mathbf{x}^{c3} = \{1, 0\}$. The choice function of cognitive node j derived using (1) is given by

$$T_j = \gamma^{c1} \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_j^{c1}|}{\alpha^{c1} + |\mathbf{w}_j^{c1}|} + \gamma^{c2} \frac{|\mathbf{w}_j^{c2}|}{\alpha^{c2} + |\mathbf{w}_j^{c2}|} + \gamma^{c3} \frac{|\mathbf{w}_j^{c3}|}{\alpha^{c3} + |\mathbf{w}_j^{c3}|}.$$

With all things equal, each cognitive node j has a uniform norm in the action and reward fields. We have

$$T_j = \gamma^{c1} \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_j^{c1}|}{\alpha^{c1} + |\mathbf{w}_j^{c1}|} + C$$

where C is a constant across all cognitive nodes. This means that the choice function T_j^c will have the highest value when each of the nonzero attributes in the template state vector $\mathbf{w}_j^{c1}$ is matched by the corresponding attributes in the input state vector $\mathbf{x}^{c1}$. In other words, $\forall i, x^{c1}(i) \geq w_j^{c1}(i)$.

Assuming a uniform distribution of the input attribute values in the state space, the probability for a state attribute to match to the corresponding nonzero weight value in the template vector encoded by a cognitive node is a constant value $p \in [0, 1]$. It follows that the probability values for all nonzero weight values of cognitive node j_1 and cognitive node j_2 to be matched by the corresponding input attributes are given by

$$Prob_{j_1} = p^{(\mathrm{N_{j_1}})} \quad \text{and} \quad Prob_{j_2} = p^{(\mathrm{N_{j_2}})}$$

where N_{j_1} and N_{j_2} are the number of nonzero attributes in the template state vectors $\mathbf{w}_{j_1}^{c1}$ and $\mathbf{w}_{j_2}^{c1}$, respectively.

Given that $|\mathbf{w}_{j_1}^{c1}| < |\mathbf{w}_{j_2}^{c1}|$, we have $N_{j_1} < N_{j_2}$, and thus

$$Prob_{j_1} > Prob_{j_2}.$$

Therefore, the choice probability of cognitive node j_1 is higher than that of cognitive nodes j_2. □

Corollary 1: Assuming all things equal, a broad propositional rule with a smaller collection of antecedent attributes than the state space $\mathbf{X}$ is likely to have a higher probability of getting selected during code competition.

However, it is also known that the match function $\mathbf{m}_J^{ck}$ derived using (2) will still have to satisfy the vigilance criterion during template matching. Therefore, the probability of cognitive node J satisfying the vigilance criterion is presented as the match probability in the following lemma.

Lemma 2 (Match Probability of Cognitive Nodes): Assuming all things equal, given two cognitive nodes j_1 and j_2 such that

$$|\mathbf{w}_{j_1}^{c1}| < |\mathbf{w}_{j_2}^{c1}|$$

the probability of the cognitive node j_1 satisfying the vigilance criterion is lower than that of cognitive node j_2.

Proof: Given that, for the same state activity vector $\mathbf{x}^{c1}$, $m_{J=j_1}^{c1}$ and $m_{J=j_2}^{c1}$ are derived as

$$m_{J=j_1}^{c1} = \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_{j_1}^{c1}|}{|\mathbf{x}^{c1}|} \quad \text{and} \quad m_{J=j_2}^{c1} = \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_{j_2}^{c1}|}{|\mathbf{x}^{c1}|}.$$

Since $|\mathbf{w}_{j_1}^{c1}| < |\mathbf{w}_{j_2}^{c1}|$, we have

$$\frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_{j_1}^{c1}|}{|\mathbf{x}^{c1}|} < \frac{|\mathbf{x}^{c1} \wedge \mathbf{w}_{j_2}^{c1}|}{|\mathbf{x}^{c1}|}.$$

Therefore, for the same state vigilance ρ^{c1}, we have

$$P\left(m_{J=j_1}^{c1} > \rho^{c1}\right) < P\left(m_{J=j_2}^{c1} > \rho^{c1}\right).$$

□

Corollary 2: Assuming all things equal, a broad propositional rule with a smaller set of antecedent attributes has a lower probability of satisfying the vigilance criterion.

Given the above analysis, a cognitive node j_{r_j} encoding broad propositional rule r_j has a higher probability of winning the code competition, but has a lower probability of satisfying the vigilance criterion. Therefore, it is inferred that using low state vigilance ρ^{c1} can allow cognitive node j_{r_j} to satisfy the state vigilance criterion after winning the code competition.

C. Action Selection Strategy

Given the use of domain knowledge, FALCON is designed to respond effectively from the onset of the learning process. Therefore, an action selection strategy is proposed with the following design principles in mind.

Design Principle 1: Given an input state, FALCON should select an existing cognitive node encoding a broad propositional rule for action selection as much as possible.

Design Principle 2: Given an input state, FALCON should select an existing cognitive node for action selection only when it is effective to this state.

As a consequence, exploration is necessary only when none of the cognitive nodes can be exploited for responding to the states. Therefore, an exploitation strategy known as greedy exploitation is proposed to maximize exploitation and minimize exploration for greater learning efficiency.

Algorithm 4 Greedy Exploitation of Existing Knowledge

```
Require: FALCON with committed cognitive nodes.
Ensure: $\rho^{c1} \rightarrow 0.0$.
1: Search category field $F_c^2$ for cognitive node $J$ such that $\mathbf{m}^{ck} \geq \rho^{ck}$.
2: if cognitive node $J$ is uncommitted then
3:     Explore for action choice $a$ using an exploration strategy.
4: else if cognitive node $J$ is committed then
5:     Perform activity readout of action field $\omega_J^{c2}$ for action choice $a$.
6: end if
7: return  action choice $a$.
```

1) Greedy Exploitation: This strategy always attempts to exploit the existing knowledge for responding to the states. With this approach, the code selection mechanism is used to search for a cognitive node that satisfies the vigilance criterion. Therefore, any cognitive node identified using the code selection mechanism is taken to be suitable for providing an action choice effective to the states.

However, it is also expected that no cognitive node can be found for certain states. This is when the action space is explored for an action choice to respond to such states. In effect, exploration of the action space is no longer regulated using some probabilistic parameters, such as ϵ and temperature T. Instead, it is performed only when none of the existing cognitive nodes can be exploited for responding to the states.

The greedy exploitation strategy outlined in Algorithm 4 is incorporated into FALCON by replacing line 3 to line 8 of Algorithm 2. Low state vigilance $\rho^{c1} \rightarrow 0.0$ is required to exploit cognitive nodes of the broad propositional rules. The insertion of the broad propositional rules increases exploitation of the committed cognitive nodes from the onset of the learning process. In this approach, FALCON explores only when none of the committed cognitive nodes can be exploited. The benefit of this approach is in the form of improved learning efficiency and reduced model complexity.

The exclusion of an action selection policy from the RL process means that the code selection mechanism of FALCON is used to balance between exploitation and exploration. The code activation and code competition procedures are, in essence, self-organizing. Therefore, it is recognized that the selection of cognitive node with an action choice effective to the states can only be controlled using the vigilance criterion. In this sense, a reward vigilance adaptation strategy is proposed to ensure the use of appropriate reward vigilance for the vigilance criterion.

2) Reward Vigilance Adaptation: The vigilance parameters ρ^{ck} for $k = \{1, 2, 3\}$ are used to select a winning cognitive node J for either action selection or learning. Individually, the state vigilance ρ^{c1} has to be low for matching generalized state patterns, the action vigilance $\rho^{c2} = 0.0$ is used for action selection and $\rho^{c2} = 1.0$ is used for learning, and the reward vigilance ρ^{c3} is used to select an effective cognitive node j that is defined as follows.

Definition 4 (Effective Cognitive Node): A cognitive node j is considered effective when it recommends action choice a to state s resulting in $\mathcal{E}_j(s, a) \geq 0.5$, and $\mathcal{E}_j(s, a)$ is derived

Algorithm 5 Adaptation of Reward Vigilance

```
Require: Updated $Q$-value $Q(s, a)$
1: Derive effectiveness $\mathcal{E}_j(s, a)$ of action choice $a$ to state $s$ using (10).
2: if $\mathcal{E}_j(s, a) \geq 0.5$ then
3:     Adapt reward vigilance $\rho^{c3}$ by applying $Q(s, a)$ to (11).
4: end if
5: return  updated reward vigilance $\rho^{c3}$.
```

using

$$\mathcal{E}_j(s, a) = \frac{\sum_p^P \kappa_p}{P} \tag{10}$$

where P is the total number of reward attributes and $\kappa_p \in \{0, 1\}$ indicates whether the action choice a is effective ($\kappa_p = 1.0$) or ineffective ($\kappa_p = 0.0$) to state s with respect to reward attribute p.

Specifically, it is proposed that reward vigilance ρ^{c3} be adapted to guarantee the selection of an effective cognitive node j using

$$\rho^{c3}(t + 1) = \min(\nu\rho^{c3}(t) + (1 - \nu)Q(s, a), \rho^{c3}(t)) \tag{11}$$

where ν is the adaptation rate of ρ^{c3} and $Q(s, a)$ is an updated estimation of the Q-value of cognitive node j with action choice a known to have $\mathcal{E}_j(s, a) \geq 0.5$.

The reward vigilance adaptation strategy, as outlined in Algorithm 5, is to be inserted after Step 11 of Algorithm 2. An effective reward vigilance can be found using the proposed reward vigilance adaptation strategy given in (12). This is because the existence of an effective reward vigilance is guaranteed using the following lemma.

Lemma 3 (Existence of Effective Reward Vigilance): Assuming Q-learning converges, given an effective cognitive node j_1 (Definition 4) recommending action choice a_1 for state s and an ineffective cognitive node j_2 recommending action choice a_2 for state s, there exists a reward vigilance ρ^{c3} such that cognitive node j_1 achieves resonance and cognitive node j_2 is reset.

Proof: Assuming Q-learning converges, $Q(s, a_1)$ and $Q(s, a_2)$ track their respective effectiveness $\mathcal{E}_{j_1}(s, a_1)$ and $\mathcal{E}_{j_2}(s, a_2)$, which are correlated to each other with respect to κ_p used in (10).

Given that cognitive node j_1 is effective and cognitive node j_2 is ineffective, we have

$$\mathcal{E}_{j_1}(s, a_1) > \mathcal{E}_{j_2}(s, a_2)$$

implying $Q(s, a_1) > Q(s, a_2)$.

By the definition of the match function seen in (2), we have

$$m_{J=j_1}^{c3} = |\mathbf{w}_{j1}^{c3}| = Q(s, a_1)$$
$$m_{J=j_2}^{c3} = |\mathbf{w}_{j2}^{c3}| = Q(s, a_2).$$

Thus, $m_{J=j_1}^{c3} > m_{J=j_2}^{c3}$.

Therefore, there exists a reward vigilance ρ^{c3} such that

$$m_{J=j_1}^{c3} \geq \rho^{c3} > m_{J=j_2}^{c3}.$$

□

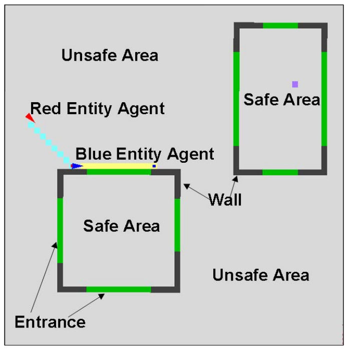
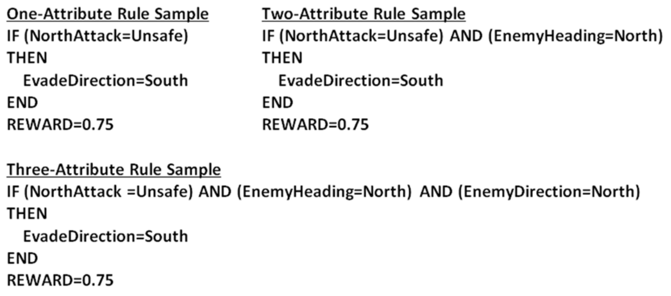


Fig. 3. PE problem domain with safe and unsafe areas.



```
One-Attribute Rule Sample
IF (NorthAttack=Unsafe)
THEN
  EvadeDirection=South
END
REWARD=0.75

Two-Attribute Rule Sample
IF (NorthAttack=Unsafe) AND (EnemyHeading=North)
THEN
  EvadeDirection=South
END
REWARD=0.75

Three-Attribute Rule Sample
IF (NorthAttack =Unsafe) AND (EnemyHeading=North)  AND (EnemyDirection=North)
THEN
  EvadeDirection=South
END
REWARD=0.75
```

Fig. 4. Samples of one-attribute, two-attribute, and three-attribute propositional rules inserted as domain knowledge.

Following Lemma 3, exploration is only required when none of the committed cognitive nodes can satisfy the vigilance criterion. Therefore, exploitation of the domain and learned knowledge is maximized leading to improved learning efficiency.

V. Pursuit-Evasion Problem Domain

The Pursuit-Evasion (PE) problem domain [29] is used here to evaluate the self-organizing neural network that integrates domain knowledge and RL. As shown in Fig. 3, there are two autonomous agents known as the blue entity agent and the red entity agent. The red entity agent is hostile toward the blue entity agent. The 2-D environment has two safe areas, where the blue entity agent will be safe from the red entity agent. The red entity agent is constantly searching for the blue entity agent. It eliminates the blue entity agent by touching it. The blue entity agent is tasked with a search mission of the areas. Therefore, it is also moving constantly. It will have to evade the red entity agent to avoid elimination.

As in [30], the pursuit strategy of the red entity agent is deterministic, while the blue entity agent progressively learns the evasive strategies to improve on its chance of evading the red entity agent over multiple training iterations. Knowledge on the desired response is communicated to the entity agent using the evaluated effect of the action choices.

A. State Space

A situation-awareness model, as defined in [31], is used as the state space of the entity agents. It comprises eight types of multivalued attributes on the enemy and the terrain. The perception layer has four types of attributes: the comprehension layer has three types of attributes and the projection layer has just one type of attribute. A state space with such a situation-awareness model can have up to around 2.86×10^{10} possible states. Further details on the state space can be found in [29].

B. Action Space

The blue entity agent learns to identify a compass direction to evade the pursuit of the red entity agent in this PE problem domain. Therefore, the action space comprises eight compass directions—north, northeast, east, southeast, south, southwest, west, and northwest—as the consequent of the decision-making task. The effect of choosing an evade direction as a response to the states is learned, and may be exploited for subsequent decision-making instances.

C. Reward Attributes

Sensory information is used to quantify the effect of action choice a on state s. The trend revealed using these reward attributes is used to derive the intermediate reward factor r using

$$r = \frac{\sum_{p}^{\mathrm{P}} \kappa_p \tau_p}{\sum_{p}^{P} \tau_p} \tag{12}$$

where $\kappa_p \in \{0, 1\}$ is the effectiveness (Definition 4) of action choice a to state s with respect to reward attribute p and τ_p is the weight that reward attribute p has on reward r. The choice of reward attributes is specific to the problem domain, and is derived based on the characteristic of the learning task. Details on the choice of reward attributes can be found in [29].

D. Domain Knowledge

Domain knowledge in the PE problem domain is represented as propositional rules. It comprises 74 broad propositional rules with either one-attribute, two-attribute, or three-attribute antecedents. Examples of such propositional rules are shown in Fig. 4. Each propositional rule comprises a set of antecedents, a consequent, and a reward value.

The antecedents of the propositional rules are defined using at least one attribute from the state space in the PE situation-awareness model. The antecedents are translated to become the state vector **S** and presented to the state field of FALCON during rule insertion.

The consequent of each propositional rule is matched to an attribute in the action space specified in Section V-B. Just a single attribute is specified for the action space of the PE problem domain. It is translated to become the action vector **A** and presented to the action field of FALCON during rule insertion.

The inserted domain knowledge is expected to allow FALCON to respond correctly in relevant states. Therefore, these propositional rules are initialized with a reward value of 0.75. The reward value is translated as reward vector **R** and presented to the reward field of FALCON during rule insertion.

E. Model Validation Criteria

1) Mission Completion Rates: Each training iteration i lasts for the duration the blue entity agent requires to complete the search mission or till it is being eliminated by the red entity agent. No timeout is included in this PE simulation. Therefore, the mission completion rates is derived using κ/w_n, where κ is the number of times the blue entity agent completes the search mission at every w_n training iterations and $\kappa \leq w_n$.

2) Code Population: This is the number of cognitive nodes at each training iteration n. The growth of the code population is expected to stagnate as the blue entity agent settles on a fixed set of evasive strategies. FALCON with a low code population is considered to have a reduced model complexity.

3) Exploitation Rates: This is the percentage at which the cognitive nodes are used for responding to the states. A higher exploitation rate is perceived as having the cognitive nodes for responding to the states. This is positive in limiting the growth of the code population. Therefore, high growth of the code population is correlated to low exploitation rates and vice versa.

F. Models for Comparisons

1) Naïve Response: An inference model only capable of random response is included for a baseline performance profile on the PE problem domain. For an action space $\mathcal{A}$ with N decision choices of which M of them are invalid to the current state, an entity agent will have $1/(N-M)$ probability of selecting any of the valid decision choices.

2) Nonadaptive Rules-Only Model: Domain knowledge is inserted into FALCON as cognitive nodes. This inserted domain knowledge is used as the only source of knowledge for action selection. RL does not apply as there is no learning. To eliminate potential bias on the selection of the domain knowledge, the direct code access mechanism of FALCON is used for the selection of action choice from among the cognitive nodes of the inserted domain knowledge. Without learning, this model responds to the states by either exploiting the inserted domain knowledge or by selecting random action choices.

3) Adapted kNN-TD(λ): The k-NN algorithm was used in a TD learning scheme during RL [17]. It is adopted in this paper with some minor adaptations. The adaptations include the use of $k = 1$ and getting each classifier to learn the mapping of state vector **S**, action vector **A**, and reward vector **A** during RL. Therefore, like cognitive node j in FALCON, each classifier cl_j has a weight template vector $\mathbf{w}_j^{ck}$, encoding the three vectors $\{\mathbf{S}, \mathbf{A}, \mathbf{R}\}$.

For action selection, the Euclidean distance $d(x^{c1}, w_j^{c1})$ between classifier cl_j and the current activity vector $\mathbf{x}^{ck}$ is determined using

$$d\left(x^{c1}, w_j^{c1}\right) = \sqrt{\sum_{i=0}^{n} \left(x_i^{c1} - w_{ji}^{c1}\right)^2} \text{ for all node j.} \tag{13}$$

In addition, this is followed by the identification of the winning classifier cl_J using

$$J = \arg\min_j d_j\left(x^{c1}, w_j^{c1}\right). \tag{14}$$

The action weight vector $\mathbf{w}_J^{c2}$ of classifier cl_J is subsequently used to derive an action choice a to state s.

For learning, bounded Q-learning and $\lambda = 0$ are used instead of the suggested eligibility traces for a fair comparison with FALCON. It learns the estimated effect $Q(s, a)$ of the chosen action choice a by identifying a winning classifier c_J using (13) and (14), which also satisfies the condition of $d(x^{c2}, w_J^{c2}) = 0$. This is to ensure that the winning node encodes the same action to be learned. A new classifier is added to learn $\{\mathbf{S}, \mathbf{A}, \mathbf{R}\}$ in the absence of a classifier that can satisfy the above selection criteria.

4) Adapted GSOM: The GSOM is included for comparison here because it is a well-known function approximator used with Q-learning [18] for similar problem types. The GSOM model is adapted to have nodes comprising $\{\mathbf{S}, \mathbf{A}, \mathbf{R}\}$. Like FALCON and kNN-TD(λ), domain knowledge is used to initialize the adapted GSOM model. During RL, the adapted GSOM alternates between two modes of operation. For action selection, (13) is used to identify the winning GSOM node J. Action choice a is then derived from the action field of winning GSOM node J.

For learning, a GSOM node j with $d(\mathbf{x}^{c2}, \mathbf{w}_j^{c2}) = 0$ is first selected. If such GSOM node j is not found, then $d(\mathbf{x}^{c1}, \mathbf{w}_j^{c1})$ is derived using (13) to find the winning GSOM node J. To determine whether $\mathbf{x}^{ck}$ is close enough to $\mathbf{w}_J^{ck}$, a distance threshold λ_d is derived using

$$\lambda_d = (1-\theta)\lambda_d + \theta d\left(\mathbf{x}^{c1}, \mathbf{w}_j^{c1}\right)$$

where θ is the adaptation rate of λ_d to $d(\mathbf{x}^{c1}, \mathbf{w}_j^{c1})$. Generalization occurs when a winning GSOM node J is used to learn $\mathbf{x}^{ck}$ when $d(\mathbf{x}^{c1}, \mathbf{w}_J^{c1}) < \lambda_d$. A new GSOM node is used to learn $\mathbf{x}^{ck}$ when $d(\mathbf{x}^{c1}, \mathbf{w}_J^{c1}) \geq \lambda_d$ or when no winning GSOM node is found.

5) Q-Learning: The standard Q-learning approach [26] is used to estimate the Q-value for every learned state-action pair. For action selection, action choice a is derived using the learned knowledge whose state field is perfectly matched to the current state s. New knowledge is learned when none of the existing knowledge is perfectly matched to state s and action choice a. The same action selection policy as the adapted kNN-TD(λ) and the adapted GSOM are used to balance between exploitation and exploration. Pruning is not used here.

VI. Experiments

Four sets of experiments are conducted using the PE and the MNT problem domains. The first set of experiments illustrates the effect of state vigilance on the use of domain knowledge. This is followed by the experiments to investigate the use of greedy exploitation with reward vigilance adaptation. The third set of experimental results compares FALCON using the proposed strategies with the naïve response system, the nonadaptive FALCON, the adapted kNN-TD(λ), and the adapted GSOM. The final set of experiments directly compares this paper with the earlier works based on the MNT problem domain.

Each experiment conducted using the PE problem domain has 500 training iterations. Each set of results is averaged using

TABLE I
PARAMETERS OF TD−FALCON AND ACTION SELECTION POLICY

FALCON for $k = \{1, 2, 3\}$		**TD Learning**	
Choice Parameters α^{ck}	$\{0.1, 0.1, 0.1\}$	Learning Rate α	0.5
Learning Rates β^{ck}	$\{1.0, 1.0, 1.0\}$	Discount Factor γ	0.1
Contribution Parameters γ^{ck}	$\{0.33, 0.33, 0.33\}$	Initial Q-Value	0.5
Vigilance ρ^{ck}	$\{\rho^{c1}, 0.0/1.0, \rho^{c3}\}$		
ρ^{c3} Adaptation Rate ν	0.95		
Confidence $(c_j(0), \zeta, \eta)$	0.5, 0.0005, 0.5		
Parameters of Action Selection Policy			
Initial ϵ value	0.0		
Window size $\mathcal{N}_w$	20 iterations		

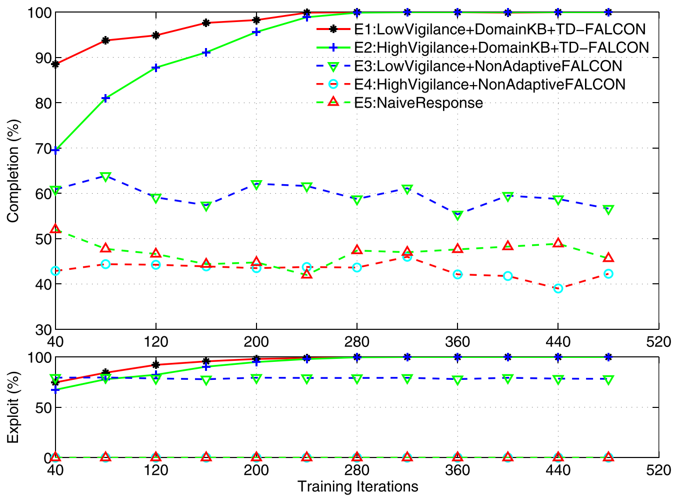


Fig. 5. Comparison of mission completion rates for the use of different state vigilance criterion.

20 runs of the same experiment. Subsequently, every 40 data points are averaged to give just 12 data points to illustrate a general trend while minimizing nonessential fluctuations. The PE experiments use $\sigma^{\text{old}} = 20$ iterations. Chosen empirically, the parameters of FALCON and the other learning processes used for the experiments based on both problem domains are presented in Table I. The same Q-learning, pruning strategy, and knowledge-based exploration strategy [29] are used for all configurations with RL.

A. State Vigilance for Domain Knowledge

Experiments conducted using high ($\rho^{c1} = 0.85$) (E2, E4) and no ($\rho^{c1} = 0.0$) (E1, E3) state vigilance are presented to illustrate its effect on learning efficiency when domain knowledge is used. FALCON inserted with domain knowledge is used in adaptive (E1, E2) and nonadaptive (E3, E4) mode, and a naïve response system (E5) is included for baseline comparison.

The comparison of mission completion rates in the top plot of Fig. 5 shows better completion rates using no state vigilance for adaptive and nonadaptive FALCON with domain knowledge. This is evident from the higher mission completion rates of E3 over E4 and of E1 over E2. From the bottom plot of Fig. 5, 0% exploitation rates is observed for E4. This signifies that none of the inserted domain knowledge can be selected when a high state vigilance is used. In effect, it responds

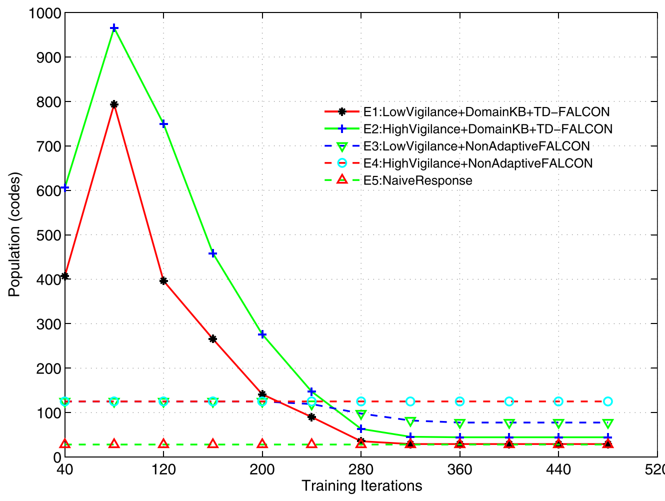


Fig. 6. Comparison of code population for the use of different state vigilance criterion.

to the states by constantly selecting random action choices. Therefore, E4 using a high state vigilance has a similar outcome as E5. In addition, E2 is much less able to fully exploit its cognitive nodes than E1. From these observations, low state vigilance is clearly preferred when there is use of domain knowledge.

Also shown in Fig. 6, high state vigilance is linked to a higher peak code population. This is evident from comparing the peak code population of E1 and E2. Even after significant pruning, E2 still has higher code population than E1. The above experimental results show that the use of no state vigilance by nonadaptive and adaptive FALCON with domain knowledge leads to better mission completion rates, exploitation rates, and lower code population. This is followed by the experiments in Section VI-B to illustrate the effect of using greedy exploitation with reward vigilance adaptation on learning efficiency.

B. Greedy Exploitation With Reward Vigilance Adaptation

The experiments presented here use FALCON to evaluate the effect of the reward vigilance adaptation strategy (E6, E8, E9) and the greedy exploitation strategy (E6, E7, E8). Comparisons are made with FALCON with no reward vigilance ($\rho^{c3} = 0.0$) (E7, E10) and FALCON with the adaptive ϵ-greedy strategy (E9, E10). E8 is a variant of E6 without pruning.

From Fig. 7, E6 [$\rho^{c3}(0) = 0.47$] is observed with the highest mission completion rate. E8 shows pruning has little effect on the mission completion rate when compared with E6. Comparison between E9 and E10 shows that adapting reward vigilance leads to more efficient convergence. Comparison between E6 and E9 shows that even higher mission completion rates are possible using the greedy exploitation strategy. Direct correlation between the mission completion rates (top plot) and the exploitation rates (bottom plot) can be observed in Fig. 7. From E7 and E10, no reward vigilance is linked to higher exploitation rates from the onset of RL. E6, E8, and E9 suggest that adapting reward vigilance can improve exploitation rates as more effective knowledge is learned.

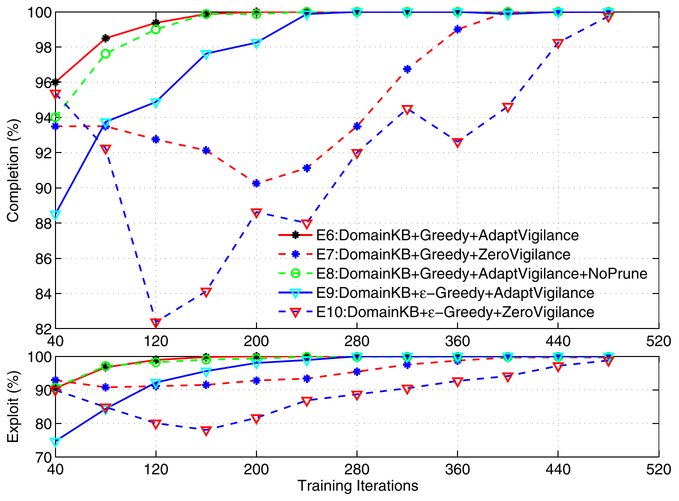


Fig. 7. Comparison of mission completion rates for using greedy exploitation and reward vigilance adaptation.

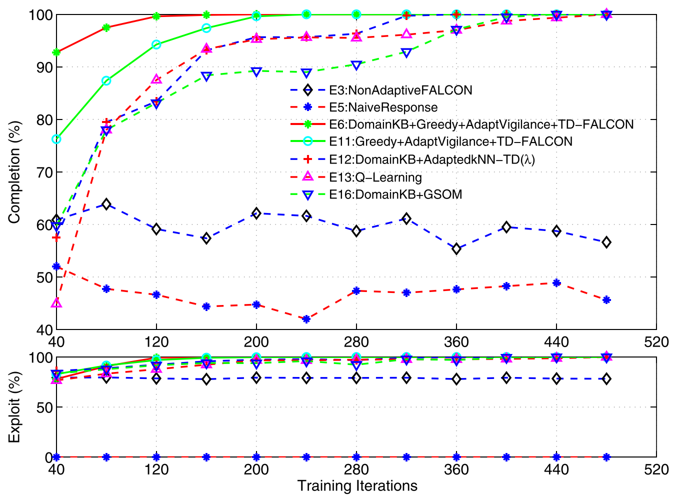


Fig. 9. Comparison of mission completion rates among the different approaches.

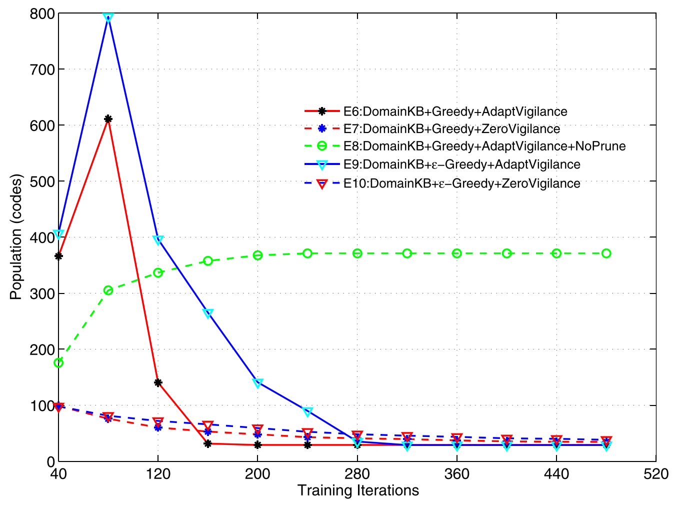


Fig. 8. Comparison of code population for using greedy exploitation and reward vigilance adaptation.

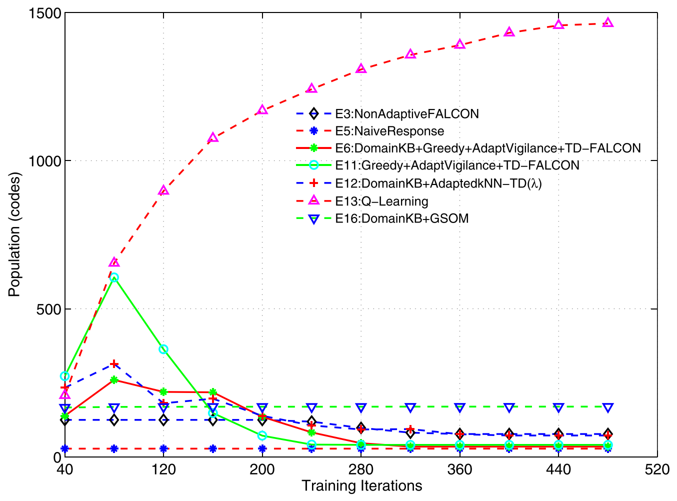


Fig. 10. Comparison of code population among the different approaches.

In contrast, the exploitation rates of E7 and E10 saturate much more gradually.

From Fig. 8, the code population of E7 and E10 are among the lowest. This is because using $\rho^{c3} = 0.0$ leads to more generalization among cognitive nodes with the same action choices ($\rho^{c2} = 1.0$). Very few new cognitive nodes are created as there is more updating of existing cognitive nodes. The higher code population of E8 accentuates the benefit of using pruning. In summary, on top of using no state vigilance, the experimental results here have illustrated significant reduction to the model complexity when reward vigilance adaptation and greedy exploitation are included.

C. Comparison With Other Function Approximators

The experiments here compare the effect of using FALCON integrated with greedy exploitation and reward vigilance adaptation (E6) with other function approximators. FALCON with no state vigilance ($\rho^{c1} = 0.0$) is used for E3, E6, and E11. Comparisons are made with E3, E5, the adapted kNN-TD(λ) (E12), the adapted GSOM (E16) (with $\theta = 0.25$), and the standard Q-learning (E13).

From the top plot of Fig. 9, illustrating just the use of domain knowledge, E3 is already better than E5 (naïve response). In comparison, E12, E13, and E16 are shown to be converging slower than E11. Consequentially, E6 has the best learning efficiency among the configurations. As can be expected, the bottom plot of Fig. 9 shows E5 with 0% exploitation rates. In contrast, E3 exploits the domain knowledge at around 80% of the time while responding randomly for all the other time. In comparison, E12, E13, and E16 require around the same duration to fully exploit the learned knowledge. In comparison, E11 is slower to achieve 100% exploitation rates than E6.

From Fig. 10, E11 is observed with a higher initial code population than E6. Without pruning, the code population of E13 continues to rise above all the others. As training progresses, E6 and E11 are able to converge on a similar code population, while those of E12 and E3 remain above them toward the end of the training process. The change in the code population is perceived as being correlated to the mission completion rates in Fig. 9. The code population for E6, E11, and E12 begins to decline as their mission completion rates saturate. The code population of E16, though higher than those of E6, E11, and E12, remains rather constant while its mission completion rates gradually improve over time. The growth of the code population of E13 slows as the mission completion rates saturate.

In terms of runtime efficiency, we compare the runtime taken by each model for each decision cycle (comprising action selection and learning) and the total runtime each model took to reach full exploitation. From our results, E13 ran at a decision cycle of 6.24 ms, and the decision cycles of E6, E12, and E16 took 0.98, 8.08, and 2.61 ms, respectively. As for the total runtime to reach 100% exploitation, E6, E12, E13, and E16 took 473.85, 3878.71, 2996.26, and 1253.89 ms, respectively. The results show that our proposed strategy (E6) is more efficient than the compared models.

D. Experiments Using MNT

In the MNT problem domain [12], an autonomous vehicle (AV) is tasked to navigate a 16×16 minefield consisting of 10 mines to a target position. Inputs to the AV include five sonar readings of mine positions and eight target bearings. The AV moves toward the destination in one of the five directions. The immediate reward is based on the absolute distance to the destination, while terminal reward of 1 is given when AV reaches the target and terminal reward of 0 is given when it either moves into a mine or fails to reach the destination in a fixed number of steps.

The scenarios, defined by the placement of destination, the mines, and the AV, are randomly generated for each run of the experiment. The performance of FALCONs with the proposed strategies (E6 and E11) in this MNT problem domain is compared with five other approaches. The experimental results are averaged using 20 runs of the experiments with 2000 training iterations each. This is followed by aggregating every 100 data points to give just 20 data points. E14 and E15 use $\rho^{ck} = \{0.25, 0.2, 0.5\}$, $\gamma = 0.5$, and $\epsilon = 0.5$. Pruning is conducted using $\sigma^{\text{old}} = 100$.

From Fig. 11, E3 shows success rates of around 82% for the five propositional rules that are also inserted in E6 and E14. E5 shows less than 20% success rates by responding randomly. E13 shows the standard Q-learning starting from around 30% success rate and achieving close to 100% after around 1100 training iterations. Like in [15], E14 and E15 attain more than 90% success rates after learning for around the same duration. In contrast, using domain knowledge based on the proposed strategies, E6 is observed with better performance than E14 in Fig. 9. The improved performance of E11 over E15 shows that the proposed strategies are also effective when no domain knowledge is used.

Fig. 12 compares the model complexity of various methods. E13 with the highest number of cognitive nodes shows the standard Q-learning to be the least efficient approach. With pruning, the code populations of E14 and E15 settle at around 500 nodes after 2000 training iterations. In contrast,

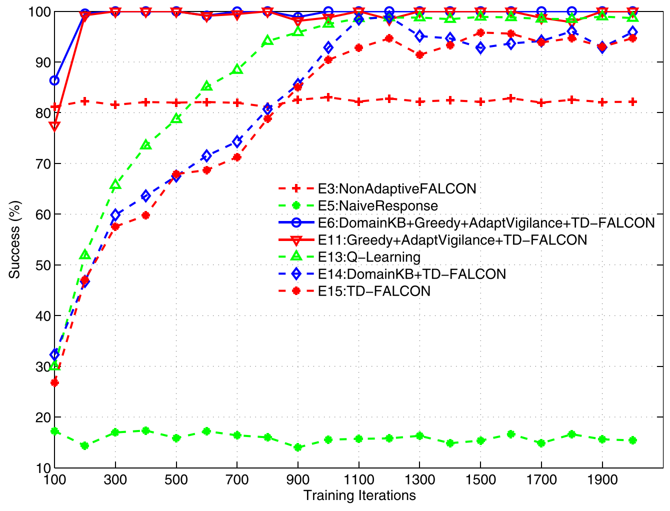


Fig. 11. Comparison of success rates as percentage of time AV navigates to its destination in the MNT problem domain.

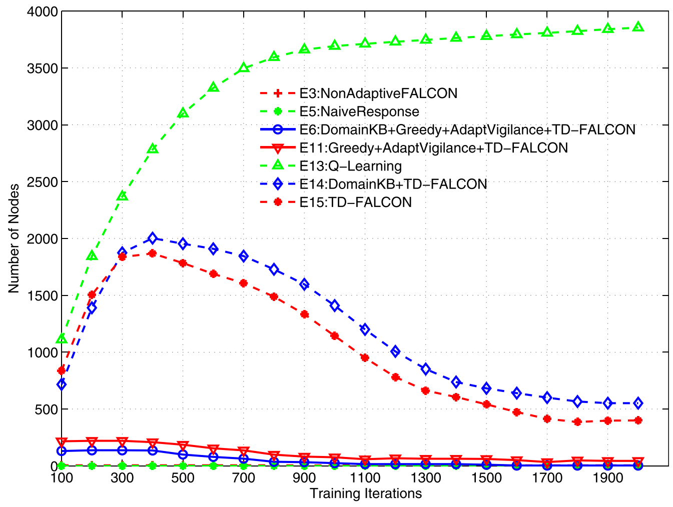


Fig. 12. Comparison of code population of the experiment configurations.

E6 and E11 are much more efficient in containing the growth of its code population. Therefore, the proposed strategies allow FALCON to learn more efficiently (shown in Fig. 12) and more effectively (shown in Fig. 11).

Following [13], comparison of timing information in the MNT problem domain is made using the runtime taken by the AV for each decision cycle and the total runtime over 2000 iterations. Empirically, E13 has a total runtime of 4 min 26 s, whereas E6, E11, E14, and E15 took around 3.91 s, 6.66 s, 2 min, and 1 min 27 s, respectively. In terms of the runtime per decision cycle, E13 took 14.21 ms. In comparison, E6, E11, E14, and E15 took 0.23, 0.42, 7.0, and 4.34 ms, respectively. Consequentially, domain knowledge in the MNT problem domain is also used more efficiently using our proposed strategies (E6).

VII. Conclusion

This paper has shown how domain knowledge can be integrated with RL using a self-organizing neural network known as TD-FALCON. We have analytically shown how the inserted domain knowledge is utilized for action selection and learning. In addition, we proposed the greedy exploitation and reward vigilance adaptation strategies to make better use of domain knowledge to improve learning efficiency. Using such an approach, exploration is triggered only when no effective cognitive node can be exploited for the states. It is shown that the appropriate cognitive nodes can be selected as the reward vigilance is adapted during RL.

To illustrate the efficacy of the proposed strategies for integrating domain knowledge with RL, experiments were conducted using the PE and MNT problem domains. Comparing with the selected models, the experiment results show that inserting domain knowledge directly into TD-FALCON using the proposed strategies improves success rates and reduces code population in these two distinct problem domains. Comparison of timing information from these two problem domains also shows the proposed strategies to be more efficient than the compared models.

This work of integrating domain knowledge and RL using a self-organizing neural network sets the framework for developing more efficient autonomous knowledge-based systems capable of continuously expanding its knowledge through real-time interaction with the environment. In our future work, we shall embark on the application of the proposed strategies in more challenging and complex real-world problem domains. Beyond the type of logical structure, domain knowledge in these problem domains is likely to be more complex and heterogeneous. By drawing inspirations from the fields of cognitive psychology and neuroscience, we aim to build self-organizing knowledge systems for addressing the issues of acquiring, managing, and retrieving such rich and diverse knowledge, possibly through the use of different types of memory representations and models [32].

References

[1] A. M. Shapiro, "How including prior knowledge as a subject variable may change outcomes of learning research," *Amer. Educ. Res. J.*, vol. 40, no. 1, pp. 159–189, 2004.

[2] A.-H. Tan, "Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 237–250, Mar. 1997.

[3] Y. S. Abu-Mostafa, "Learning from hints in neural networks," *J. Complex.*, vol. 6, no. 2, pp. 192–198, 1990.

[4] M. Pazzani and D. Kibler, "The utility of knowledge in inductive learning," *Mach. Learn.*, vol. 9, no. 1, pp. 57–94, 1992.

[5] T. M. Mitchell and S. Thrun, "Explanation-based neural network learning for robot control," in *Advances in Neural Information Processing Systems 5*. San Mateo, CA, USA: Morgan Kaufmann, 1993, pp. 287–294.

[6] G. G. Towell and J. W. Shavlik, "Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules," in *Advances in Neural Information Processing Systems 4*. San Mateo, CA, USA: Morgan Kaufmann, 1992, pp. 977–984.

[7] L.-M. Fu, "Knowledge-based connectionism for revising domain theories," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 1, pp. 173–182, Jan./Feb. 1993.

[8] C. H. C. Ribeiro, "Embedding *a priori* knowledge in reinforcement learning," *J. Intell. Robot. Syst.*, vol. 21, no. 1, pp. 51–71, 1998.

[9] R. Schoknecht, M. Spott, and M. Riedmiller, "Fynesse: An architecture for integrating prior knowledge in autonomously learning agents," *Soft Comput.*, vol. 8, no. 6, pp. 397–408, 2004.

[10] G. Hailu and G. Sommer, "Integrating symbolic knowledge in reinforcement learning," in *Proc. Int. Conf. Syst., Man, Cybern.*, vol. 2. Oct. 1998, pp. 1491–1496.

[11] D. Shapiro, P. Langley, and R. Shachter, "Using background knowledge to speed reinforcement learning in physical agents," in *Proc. Int. Conf. Auto. Agents*, May 2001, pp. 254–261.

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

[13] A.-H. Tan, N. Lu, and X. Dan, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 230–244, Feb. 2008.

[14] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vis., Graph., Image Process.*, vol. 37, no. 1, pp. 54–115, 1987.

[15] T.-H. Teng, Z.-M. Tan, and A.-H. Tan, "Self-organizing neural models integrating rules and reinforcement learning," in *Proc. IEEE IJCNN*, Jun. 2008, pp. 3770–3777.

[16] T.-H. Teng and A.-H. Tan, "Cognitive agents integrating rules and reinforcement learning for context-aware decision support," in *Proc. IAT*, Dec. 2008, pp. 318–321.

[17] J. A. Martín H., J. de Lope, and D. Maravall, "The *k*NN-TD reinforcement learning algorithm," in *Methods and Models in Artificial and Natural Computation. A Homage to Professor Mira's Scientific Legacy* (Lecture Notes in Computer Science), vol. 5601. Berlin, Germany: Springer-Verlag, 2009, pp. 305–314.

[18] H. Montazeri, S. Moradi, and R. Safabakhsh, "Continuous state/action reinforcement learning: A growing self-organizing map approach," *Neurocomputing*, vol. 74, no. 7, pp. 1069–1082, 2011.

[19] P. Niyogi, F. Girosi, and T. Poggio, "Incorporating prior information in machine learning by creating virtual examples," *Proc. IEEE*, vol. 86, no. 11, pp. 2196–2209, Nov. 1998.

[20] I. Guyon, A. Saffari, G. Dror, and G. Cawley, "Agnostic learning vs. prior knowledge challenge," in *Proc. IJCNN*, Orlando, FL, USA, Aug. 2007, pp. 829–834.

[21] H. Langseth and T. D. Nielsen, "Fusion of domain knowledge with data for structural learning in object oriented domains," *J. Mach. Learn. Res.*, vol. 4, pp. 339–368, Dec. 2003.

[22] R. Mahony and R. Williamson, "Prior knowledge and preferential structures in gradient descent learning algorithms," *J. Mach. Learn. Res.*, vol. 1, no. 4, pp. 311–355, 2001.

[23] A. R. Masegosa and S. Moral, "An interactive approach for Bayesian network learning using domain/expert knowledge," *Int. J. Approx. Reasoning*, vol. 54, no. 8, pp. 1168–1181, 2013.

[24] F. Lauer and G. Bloch, "Incorporating prior knowledge in support vector machines for classification: A review," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1578–1594, 2008.

[25] C. Dugas, Y. Bengio, F. Belisle, C. Nadeau, and R. Garcia, "Incorporating functional knowledge in neural networks," *J. Mach. Learn. Res.*, vol. 10, pp. 1239–1262, Dec. 2009.

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

[27] T.-H. Teng, A.-H. Tan, and Y.-S. Tan, "Self-regulating action exploration in reinforcement learning," *Proc. Comput. Sci.*, vol. 13, pp. 62–74, Oct. 2012.

[28] A.-H. Tan, "Direct code access in self-organizing neural networks for reinforcement learning," in *Proc. IJCAI*, Jan. 2007, pp. 1071–1076.

[29] T.-H. Teng and A.-H. Tan, "Knowledge-based exploration for reinforcement learning in self-organizing neural networks," in *Proc. IAT*, Dec. 2012, pp. 332–339.

[30] S. Ficici and J. Pollack, "Statistical reasoning strategies in the pursuit and evasion domain," in *Advances in Artificial Life* (Lecture Notes in Computer Science), vol. 1674, D. Floreano, J.-D. Nicoud, and F. Mondada, Eds. New York, NY, USA: Springer-Verlag, 1999, pp. 79–88.

[31] M. R. Endsley, "Situation awareness: Progress and directions," in *A Cognitive Approach to Situation Awareness: Theory and Application*. Aldershot, U.K.: Ashgate Pub., 2004, ch. 17, pp. 317–341.

[32] E. Tulving, "How many memory systems are there?" *Amer. Psychol.*, vol. 40, no. 4, pp. 385–398, 1985.

Teck-Hou Teng (M'12) received the B.E. (Hons.) degree in computer engineering and the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2003 and 2013, respectively, with the thesis titled Cognitive Information System for Context-Aware Decision Support. He was with the Modeling & Simulation Systems Team, Temasek Laboratories, NTU, from 2003 to 2006, where he was involved in the behavioral modeling of CGF using FSM. Previously, he was with the University of Salford, Lancashire, U.K., where he was involved in the control mechanism of a dexterous robotics arm. He was then with INRIA Rhone-Alpes, Grenoble, France, for six months, where he was involved in a simulator for an electric vehicle in 2002. During the Ph.D. candidature, he had collaborated with academic and industrial partners to investigate the use of adaptive CGF for training and simulation. He has authored on topics such as cognitive information system, reinforcement learning, and the use of domain knowledge by self-organizing neural networks.

Ah-Hwee Tan (SM'04) received the B.Sc. (Hons.) and M.Sc. degrees in computer and information science from the National University of Singapore, Singapore, in 1989 and 1991, respectively, and the Ph.D. degree in cognitive and neural systems from Boston University, Boston, MA, USA, in 1994. He is currently an Associate Professor with the School of Computer Engineering, Nanyang Technological University (NTU), where he was the Head of Division and the Founding Director of the Emerging Research Laboratory, a center for incubating new interdisciplinary research initiatives. Prior to joining NTU, he was a Research Manager with the Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore, where he was spearheading the Text Mining and Intelligent Agents research programs. He has authored more than 160 technical papers in major international journals and conferences of his fields, and six edited books and proceeding volumes. He holds two U.S. patents, five Singapore patents, and has spearheaded several A*STAR projects in commercializing a suite of document analysis and text mining technologies. His current research interests include cognitive and neural systems, brain-inspired intelligent agents, machine learning, knowledge discovery, and text mining. Dr. Tan has served as an Associate Editor/Editorial Board Member of several international journals, including the IEEE ACCESS and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS SYSTEMS.

Jacek M. Zurada (LF'14) received the Ph.D. degree from the Gdańsk Institute of Technology, Gdańsk, Poland. He currently serves as a Professor of Electrical and Computer Engineering with the University of Louisville, Louisville, KY, USA. He has authored and co-authored several books and over 370 papers in computational intelligence, neural networks, machine learning, logic rule extraction, and bioinformatics, and delivered numerous presentations and seminars throughout the world. His work has been cited over 8000 times. Dr. Zurada currently serves as an IEEE Vice President, Technical Activities (TAB Chair), and the Chair of the IEEE TAB Management Committee. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON NEURAL NETWORKS from 1997 to 2003, an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and was an Editorial Board Member of the PROCEEDINGS OF THE IEEE. From 2004 to 2005, he was the President of the IEEE Computational Intelligence Society. He is an Associate Editor of *Neurocomputing*, *Neural Networks*, and several other international journals. He holds the title of a Professor in Poland, is a member of the Polish Academy of Sciences, and has been awarded numerous awards, including honorary professorships of four Chinese universities, including Sichuan University, Chengdu, China. He is a Board Member of the IEEE Computational Intelligence Society and the International Joint Conference on Neural Networks.