12-2012

Knowledge-based exploration for reinforcement learning in self-organizing neural networks

Teck-Hou TENG

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

Knowledge-based Exploration for Reinforcement Learning in Self-Organizing Neural Networks

Teck-Hou Teng, Ah-Hwee Tan
School of Computer Engineering, Nanyang Technological University, Singapore 639798
Email: thteng, asahtan@ntu.edu.sg

***Abstract*—Exploration is necessary during reinforcement learning to discover new solutions in a given problem space. Most reinforcement learning systems, however, adopt a simple strategy, by randomly selecting an action among all the available actions. This paper proposes a novel exploration strategy, known as Knowledge-based Exploration, for guiding the exploration of a family of self-organizing neural networks in reinforcement learning. Specifically, exploration is directed towards unexplored and favorable action choices while steering away from those negative action choices that are likely to fail. This is achieved by using the learned knowledge of the agent to identify prior action choices leading to low Q-values in similar situations. Consequently, the agent is expected to learn the right solutions in a shorter time, improving overall learning efficiency. Using a Pursuit-Evasion problem domain, we evaluate the efficacy of the knowledge-based exploration strategy, in terms of task performance, rate of learning and model complexity. Comparison with random exploration and three other heuristic-based directed exploration strategies show that Knowledge-based Exploration is significantly more effective and robust for reinforcement learning in real time.**

***Keywords*-Reinforcement Learning, Self-Organizing Neural Network, Directed Exploration, Rule-Based System**

I. Introduction

Exploration is a necessary step during reinforcement learning (RL) for the discovery of new solutions in a given problem space [1]. During RL, an action choice is picked to explore its effect on a given situation. Therefore, for a given problem domain, the choice of exploration strategy has a role in the learning efficiency of the agent.

There are undirected and directed exploration strategies [2]. Exploration is undirected when knowledge is not used for the selection of action choices for exploration. The ϵ-greedy, Boltzmann distribution and softmax are some examples of undirected exploration strategies. Directed exploration uses some knowledge to direct subsequent exploration. Examples of recent directed exploration strategies are the BIMM models [3], the reuse of policy [4] and the heuristics exploration strategies [5]. These methods direct exploration by tracking recently explored actions in their own ways. However, they do not make use of learned knowledge of the learning agents.

Taking a different view from the other directed exploration strategies, this paper proposes a novel exploration strategy known as *Knowledge-Based Exploration*. Specifically, it aims to improve the efficiency of learning by making use of the learned knowledge to avoid re-exploration of the action choices known to lead to undesirable outcome in similar situations. To this end, the learned knowledge is dichotomized into positive and negative chunks. While *positive chunks* refer to the knowledge of action choices that lead to desirable outcomes, *negative chunks* refer to the knowledge of action choices known to produce undesirable outcomes (indicated by low Q-values) for a given situation [6]. Consequentially, the Knowledge-Based Exploration strategy directs the exploration of new action choices away from those action choices encoded by negative chunks for similar situations.

In this paper, the Knowledge-based Exploration Strategy is illustrated using a self-organizing neural network known as Temporal Difference - Fusion Architecture for Learning and Cognition (TD-FALCON) [7]. It is a 3-channel fusion Adaptive Resonance Theory (ART) network [8] that incorporates temporal difference methods [9] into the ART models [10] for RL. Using the knowledge encoded by the cognitive nodes in TD-FALCON, positive and negative chunks can be identified readily. Working in tandem, the Knowledge-Based Exploration strategy leverages on the learned knowledge for guiding the exploration during RL. Experiments are conducted using a complex pursuit-evasion (PE) problem domain [11] to evaluate the efficacy of the proposed Knowledge-based Exploration Strategy in terms of task performance, learning speed and resource utilization. Comparison with the baseline random exploration and three other heuristic-based exploration strategies have shown that the Knowledge-based Exploration is significantly more effective and robust for RL in real time.

Having introduced and motivated our work, we shall then provide a survey of the related work in Section II. This is followed by a summarized presentation of TD-FALCON in Section III. Details on how knowledge can be inserted and pruned are presented in Section IV. The proposed Knowledge-Based Exploration strategy is detailed in Section V. The PE problem domain and the performance measures are introduced in Section VI. The experiments and the results are presented and analysed in Section VII. The final section concludes with a discussion of the key results and provides a brief description of the future work.

II. Related Work

During reinforcement learning, knowledge is discovered and used through exploration and exploitation respectively. The balance between these two phases is managed using an action selection policy such as the ϵ-greedy and Boltzmann distribution. However, the selection of action choices during exploitation and exploration is independent of the action selection policy.

Exploration of action choices during RL can either be undirected or directed [2]. The absence of the use of exploration-specific information characterizes the undirected exploration strategy where action choices are randomly selected with uniform probability for exploration. The choice of different action selection policies result in different probability distribution of picking random action choices for exploration. Such a random exploration strategy with the ϵ-greedy method is compared with our proposed strategy in the experiments.

In contrast, directed exploration makes use of exploration-specific information to select action choices for exploration. Recent works on directed exploration includes the BIMM model that uses neural network to learn pre-existing knowledge to direct exploration of state space [3]. However, their choice of pre-existing knowledge is specified externally. In another work, policies learned from different tasks are re-used for the learning of new task in the same domain [4]. The presence of learned policies is assumed and a supervisor is required to identify similar policies for reuse. Without any of these assumptions, the proposed Knowledge-based Exploration Strategy directs exploration using knowledge learned continuously.

Three heuristic search-based exploration strategies are proposed for RL [5]. All three strategies, namely the Neighborhood Search-based Exploration, the Simulated Annealing-based Exploration and the Tabu Search-based Exploration track the use of action choices for each state and directs exploration towards the most unvisited actions for the state. These exploration strategies differ in the tracking and selection of the action choices. Comparisons are made with these exploration strategies because they are most compatible to the general approach adopted in this work.

Well suited to the agent-based approach, FALCON integrated with rules is used to implement cognitive agent to provide context-aware decision support [12]. Collaborating with a world-renowned simulator manufacturer, FALCON is used to implement computer-generated force (CGF) to evolve 1-v-1 air combat maneuvering strategies against another CGF [13] and also the human pilots [14] separately.

III. The Reinforcement Learning Model

Although the proposed Knowledge-Based Exploration may be applicable to the other RL systems, this paper illustrates its use through a self-organizing neural network model, called TD-FALCON [7]. By learning multi-dimensional mappings across states, actions and values in an online and incremental manner, TD-FALCON enables RL of both value and action polices in real time.

A. Structure and Operating Modes

Structurally, the FALCON network [15] employs a 3-channel architecture (Fig. 1), comprising of 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 $\mathbf{S}$, 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 knowledge layer has the category field F_2^c for storing the committed and uncommitted cognitive nodes. Each cognitive node j has three fields of weights $\mathbf{w}_j^{ck}$ for $k = 1, \ldots, 3$.

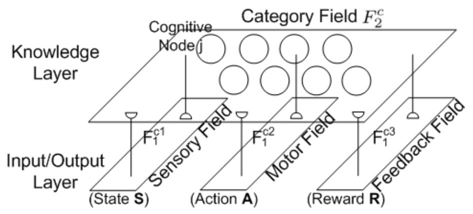

Fig. 1. The FALCON Architecture.

The FALCON network has three modes of operations - INSERT, PERFORM and LEARN. The modes of operation determine how the various parameters should be used. Relevant to this work, some details on how *prior* knowledge can be inserted and prune are provided in Section IV. In the context of this work, it is sufficient to just outline the FALCON generic network dynamics using Algorithm 1.

Algorithm 1 FALCON Generic Network Dynamics

```
Require: Activity vectors $\mathbf{x}^{ck}$ and all weights vector $\mathbf{w}_j^{ck}$
 1: for each $F_2^c$ node $j$ do
 2:     Code Activation: Derive choice function $T_j^c$ using
            $T_j^c = \sum_{k=1}^{3} \gamma^{ck} \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_j^{ck}|}{\alpha^{ck} + |\mathbf{w}_j^{ck}|}$
        where the fuzzy AND operation $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, the norm
        $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$, $\alpha^{ck} \in [0,1]$ is
        the choice parameters, $\gamma^{ck} \in [0,1]$ is the contribution parameters and
        $k = 1, \ldots, 3$
 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: Check whether the match functions $m_J^{ck}$ of
        cognitive node $J$ meet the vigilance criterion
            $m_J^{ck} = \frac{|\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck}|}{|\mathbf{x}^{ck}|} \geq \rho^{ck}$
        where $\rho^{ck} \in [0,1]$ for $k = 1, \ldots, 3$ are the vigilance parameters
 7:     if vigilance criterion is satisfied then
 8:         Resonance State 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: $m_J^{ck} = 0.0$
12:     end if
13: until Resonance State is attained
14: if operating in LEARN/INSERT mode then
15:     Template Learning: modify weight vector $\mathbf{w}_J^{ck}$ using
            $\mathbf{w}_J^{ck(\text{new})} = (1 - \beta^{ck})\mathbf{w}_J^{ck(\text{old})} + \beta^{ck}(\mathbf{x}^{ck} \wedge \mathbf{w}_J^{ck(\text{old})})$
        where $\beta^{ck} \in [0,1]$ is the learning rate
16: else if operating in PERFORM mode then
17:     Activity Readout: Read out the action vector $\mathbf{A}$ of cognitive node $J$
        using
            $\mathbf{x}^{c2(new)} = \mathbf{x}^{c2(old)} \wedge \mathbf{w}_J^{c2}$
        Decode $\mathbf{x}^{c2(new)}$ to derive recommended action choice $a$
18: end if
```

B. Incorporating Temporal Difference Method

Outlined in Algorithm 2, TD-FALCON [7] incorporates Temporal Difference (TD) methods to estimate and learn value function $Q(s,a)$ of state-action pair that indicates the goodness of taking action choice a in state s. Upon receiving a feedback from the environment after performing the action, a TD formula is used to estimate the Q-value for performing the chosen action in the previous state. The estimated Q-value is used as the teaching signal to TD-FALCON to learn the association of state s and action choice a.

Algorithm 2 The TD-FALCON Algorithm

```
1: Initialize FALCON
2: Sense the environment and formulate a state representation $s$
3: Use *Action Selection Policy* to decide between **Exploration** and **Exploitation**
4: **if** Exploration **then**
5:     Use *Exploration Strategy* to select an action choice from action space
6: **else if** Exploitation **then**
7:     Use *Direct Code Access* [16] to select an action choice from existing knowledge
8: **end if**
9: Use action choice $a$ on state $s$ for state $s'$
10: Evaluate effect of action choice $a$ to derive a reward $r$ from the environment
11: Estimate the $Q$-value function $Q(s,a)$ following a temporal difference formula given by $\Delta Q(s,a) = \alpha TD_{err}$
12: Present $\mathbf{S}$, $\mathbf{A}$ and $\mathbf{R}$ for **Learning**
13: Update the current state $s = s'$
14: Repeat from Step 2 until $s$ is a terminal state
```

Iterative Value Estimation: A value function based on a temporal difference method known as Bounded Q-Learning [9] is used to iteratively estimate the value of applying action choice a to situation s. The estimated Q-value $Q(s,a)$ is learned by TD-FALCON during RL. The temporal difference of the value function is iteratively estimated using

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

where $\alpha \in [0,1]$ is the learning parameter, the term $(1 - Q(s,a))$ allows the adjustment of Q-values to be self-scaling in such a way that it will not be increased beyond 1.0 and TD_{err} is the temporal error term derived using

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

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' and r is either the intermediate or terminal reward.

C. Action Selection Policy

During RL, an agent alternates between exploitation of learned knowledge and the exploration of solution space for more effective knowledge. In this work, TD-FALCON employs the ϵ-greedy method, which selects action choices with the highest value with a probability of $1-\epsilon$ and takes a random action with a probability of ϵ [17]. In practice, it is beneficial to have a higher ϵ value to encourage more exploration in the initial stage and a lower ϵ value to optimize the performance by exploiting the learned knowledge in the later stage. Therefore, a linear ϵ-decay policy is adopted to gradually decay ϵ by a fixed decay rate θ over time.

IV. Knowledge Insertion and Pruning

Symbol-based propositional rules are inserted into TD-FALCON as an initial set of knowledge. The learned knowledge found to be ineffective for the task is pruned from TD-FALCON. The mechanism for inserting and pruning knowledge in TD-FALCON are presented in this section.

A. Knowledge Insertion

As the knowledge structure of TD-FALCON is compatible with the structure of generalized modus ponens, *prior* knowledge in the form of propositional rules can be inserted into TD-FALCON [18] before learning. Given a rule of the form

IF antecedents THEN consequents FOR reward,

the antecedents are translated into state vector $\mathbf{S}$ and the consequents are translated into action vector $\mathbf{A}$. The reward vector $\mathbf{R} = \{r, 1-r\}$ encodes an estimated Q-value of the inserted rule. The state vector $\mathbf{S}$, action vector $\mathbf{A}$ and reward vector $\mathbf{R}$ are then inserted into TD-FALCON using the Generic Network Dynamics outlined in Algorithm 1.

Algorithm 3 The Rule Insertion Algorithm

```
1: Initialize TD-FALCON
2: Initialize $\rho^{ck} \leftarrow 1.0$
3: **for** each propositional rule **do**
4:     Translate each component of the propositional rule into the vector format
5:         *antecedents* is translated as state vector $\mathbf{S}$
6:         *consequents* is translated as action vector $\mathbf{A}$
7:         *reward* is translated as reward vector $\mathbf{R}$
8:     Present $\mathbf{S}$, $\mathbf{A}$ and $\mathbf{R}$ to TD-FALCON for learning
9: **end for**
```

Indicated in Algorithm 3, rules are inserted using vigilance parameters $\rho^{ck} = 1.0$ for $k = \{1,2,3\}$. This is to ensure that only identical set of state, action and reward vectors are grouped into the same cognitive node. Thus, each inserted rule leads to a committed cognitive node encoding the $\{\mathbf{S}, \mathbf{A}, \mathbf{R}\}$ tuple as its weight templates. Hence, there can be as many cognitive nodes as the number of inserted rules. The TD method is used to refine these inserted rules during RL.

B. Knowledge Pruning

During RL, the residual effect of exploration is the learning of knowledge that turns out to be ineffective or spurious patterns. Action selection and learning become inefficient when these irrelevant cognitive nodes are not pruned. Therefore, such knowledge needs to be pruned for more efficient operation of TD-FALCON. A confidence-based pruning strategy similar to the one proposed in [15] is adopted to prune these 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 $\sigma_j(0)$. The confidence level c_j of cognitive node j picked for action selection and updating is reinforced using

$$c_j^{new} = c_j^{old} + \eta(1 - c_j^{old}),$$

where η is the reinforcement rate of the confidence level for all cognitive nodes. After each training iteration, the confidence level of all cognitive nodes are decayed using

$$c_j^{new} = c_j^{old} - \zeta c_j^{old}$$

where ζ is the decay rate of the confidence level for all cognitive nodes. At the same time, the age σ_j of cognitive node j is also incremented.

The age attribute σ_j of cognitive node j prevents it from being pruned when $\sigma_j = \sigma_j(0)$, $c_j = c_j(0)$ and $c_j < c^{rec}$ where c^{rec} is the recommended confidence threshold. A cognitive node j is pruned only when $c_j < c^{rec}$ and $\sigma_j \geq \sigma^{old}$ where σ^{old} is the old age threshold.

V. Knowledge-Based Exploration

Prior to this work, TD-FALCON adopts a random exploration strategy, which randomly selects an action choice from the pool of available actions. The key limitation of such undirected exploration strategy is that it excludes any consideration of the action choice during exploration. While the assumption that a suitable action choice will show up eventually is statistically valid, repeatedly selecting action choices known to be ineffective is highly inefficient and a waste of exploration opportunities.

To address these shortcomings, this paper proposes a directed exploration strategy known as Knowledge-Based Exploration. It leverages on the learned knowledge of the agent to screen action choices for exploration. As a result, the efficiency of exploration can be improved by selecting action choices from the reduced action space. The details of the proposed strategy are presented in the following sections.

A. Knowledge Representation in TD-FALCON

Each cognitive node in TD-FALCON represents an associative mapping across the state, action, and reward spaces. Given a cognitive node j, the encoded weight template vectors for the three pattern channels, namely $\{\mathbf{w}_j^{c1}, \mathbf{w}_j^{c2}, \mathbf{w}_j^{c3}\}$ can be interpreted as a *chunk* $\mathcal{C}(\mathbf{S}_j, \mathbf{A}_j, \mathbf{R}_j)$, where $\mathbf{S}_j$ represents a set of states characterized by the weight template vector $\mathbf{w}_j^{c1}$, $\mathbf{A}_j$ represents a set of actions characterized by the weight template vector $\mathbf{w}_j^{c2}$, and $\mathbf{R}_j$ represents a range of Q-values characterized by the weight template vector $\mathbf{w}_j^{c3}$. Specifically, through the learning of complement-coded input reward vectors, $\mathbf{R}_j = \{L_j, U_j\}$, where $L_j = 1 - \mathbf{w}_{j1}^{c3}$ and $U_j = \mathbf{w}_{j1}^{c3}$ are the lower and upper bounds of the Q-values.

Based on the Q-values encoded in the chunks, each such learned knowledge chunk $\mathcal{C}(\mathbf{S}_j, \mathbf{A}_j, \mathbf{R}_j)$ can be dichotomized into either positive or negative chunks using the *desirability discriminant* parameters $\{\mu^-, \mu^+\}$ as follows.

Definition: A chunk $\mathcal{C}(\mathbf{S}_j, \mathbf{A}_j, \mathbf{R}_j)$ is a *positive chunk* if the lower bound of its Q-values (for performing an action $a \in \mathbf{A}_j$ in situation $s \in \mathbf{S}_j$) L_j is above μ^+.

Definition: A chunk $\mathcal{C}(\mathbf{S}_j, \mathbf{A}_j, \mathbf{R}_j)$ is a *negative chunk* if the upper bound of its Q-values (for performing an action $a \in \mathbf{A}_j$ in situation $s \in \mathbf{S}_j$) U_j is below μ^-.

For a given situation, TD-FALCON knows the likely outcome from performing an action choice by differentiating between positive and negative chunks. Specifically, the identification of a negative chunk encoding the given pair of current state and action allows the agent to keep specific action choice out of the pool of available actions. This will help to avoid applying an action choice a already known to be undesirable.

B. Searching for Positive/Negative Chunks

For state s and action choice a, a parallel search of applicable chunks can be performed across all the cognitive nodes. Specifically, the state vector $\mathbf{S}$ and the action vector $\mathbf{A}$ representing the state s and action a respectively are used to search for the most similar chunk. The search process is implemented using Line 1-13 of the FALCON Generic Networks Dynamics.

High state and action vigilance parameters are used to search for a chunk that encodes a similar state s' with the specific action a. The Q-value is not considered during the search by using zero reward vigilance parameter. The Q-value of the selected chunk is then evaluated to determine its polarity. Specifically, a selected chunk with a range of Q-values above μ^+ is a positive chunk while one with a range of Q-value below μ^- is a negative chunk. The following lemma shows that the fuzzy ART search procedure is guaranteed to find the positive or negative chunk if there exists one.

Lemma 1 - **Guaranteed Search of Chunks:** Suppose a chunk $\mathcal{C}(\mathbf{S}_{j^*}, \mathbf{A}_{j^*}, \mathbf{R}_{j^*})$ exists for a given action choice a in state s, the fuzzy ART code search procedure will enter into a resonance state with the input state vector $\mathbf{S}$ and action vector $\mathbf{A}$ encoding the state s and action choice a respectively.

Proof - Assuming a chunk $\mathcal{C}(\mathbf{S}_{j^*}, \mathbf{A}_{j^*}, \mathbf{R}_{j^*})$ exists for a given action choice a in state s. And suppose the winning cognitive node J_1 selected by the fuzzy ART code activation and competition processes is not a chunk for state s and action a, i.e., $J_1 \neq j^*$.

This means, either s is dissimilar to $\mathbf{w}_{J_1}^{c1}$ such that $m_{J_1}^{c1} < \rho^{c1}$ and $0.75 \leq \rho^{c1} \leq 1.0$ or a is dissimilar to $\mathbf{w}_{J_1}^{c2}$ such that $m_{J_1}^{c2} < \rho^{c2}$ and $\rho^{c2} = 1.0$

Any of such situation will result in the failure of template matching, leading to a reset of winning cognitive code J_1, i.e., $T_{J_1}^c = 0$. Consequently, a new search for another cognitive node ensues until a resonance state, i.e., $m^{ck} \geq \rho^{ck}$, is achieved using another committed cognitive node J_2 such that $J_2 \neq J_1$ and $J_2 = j^*$. ■

C. The Knowledge-Based Exploration Algorithm

The Knowledge-Based Exploration strategy outlined in Algorithm 4 limits the action choice during exploration to the set of unexplored action choices $\mathcal{A}^u$ and the set of positive action choices $\mathcal{A}^+$. While *unexplored actions* refers to those not encoded by any existing chunks in TD-FALCON for a given state, *positive actions* are those encoded by positive chunks with similar states. The choice of an *unexplored actions* has the potential benefit of discovering new positive action choices but at a potential cost of discovering negative action choices. The choice of a *positive action* has the potential benefit of reaffirming its effectiveness but sacrifices the chance of trying out new actions. Both operations are considered to be beneficial for fulfilling the goals of RL.

Given the current state s, the Knowledge-Based Exploration strategy loops through all possible actions in the action space $\mathcal{A}$ and partitions them into the set of positive actions $\mathcal{A}^+$, the set of negative actions $\mathcal{A}^-$ and the set of unexplored actions $\mathcal{A}^u$. Thereafter, Knowledge-Based Exploration selects an action choice randomly from the reduced action space $\mathcal{A}^r$ given by $\mathcal{A}^+ \cup \mathcal{A}^u$ for exploration. This means the probability of selecting an unexplored action choice or a positive action choice is in direct proportion to the number of unexplored action choices and the positive action choices.

Algorithm 4 Knowledge-based Exploration of Action Space

```
Require: State vector $\mathbf{S}$ representing the state $s$
Require: Reward vector $\mathbf{R} = \{1, 1\}$
 1: for each action choice $a$ in the action space $\mathcal{A}$ do
 2:     Encode action choice $a$ as an action vector $\mathbf{A}$
 3:     Present $\mathbf{S}$ and $\mathbf{A}$ to TD-FALCON for searching of chunks
 4:     if a positive chunk is found then
 5:         action $a$ is a positive action, i.e., $a \in \mathcal{A}^+$
 6:     else if a negative chunk is found then
 7:         action $a$ is a negative action, i.e., $a \in \mathcal{A}^-$
 8:     else
 9:         action $a$ is unexplored, i.e., $a \in \mathcal{A}^u$
10:     end if
11: end for
12: Create a reduced action space $\mathcal{A}^r \equiv \mathcal{A}^+ \cup \mathcal{A}^u$
13: Randomly select an action choice $a$ from $\mathcal{A}^r$ for exploration
14: return action choice $a$
```

VI. The Pursuit-Evasion Problem Domain

The PE problem domain is a popular choice in the field of game theory as well as machine learning [19]. There are works just on evolving either the pursuer [13], [20] or the evaders [21] as well as the co-evolution of strategies for both types of agent [22]. The PE problem in this work is designed to be complex such that the learning task is non-trivial. Hence, the complexity of this problem domain necessitates the use of a Situation-Awareness Model and a combinative reward scheme with multiple reward attributes.

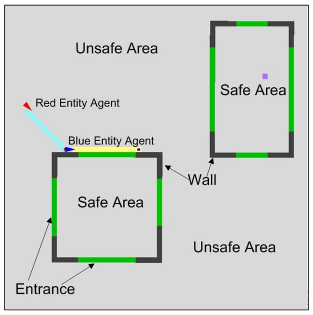

Fig. 2. The $2D$ Grid-Based Environment for the PE problem domain.

As illustrated in Fig. 2, there are two autonomous agents known as the Blue entity agent and the Red entity agent. The Red entity agent is hostile towards the Blue entity agent. The two-dimensional 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 contacting 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.

The pursuit strategy of the Red entity agent is deterministic while the Blue entity agent learns the evasive strategies to improve on its chance of evading the Red entity agent progressively. Knowledge on the desired response is implicitly communicated to the entity agent using the Situation-Awareness Model and the evaluated effect of the action choices.

A. The State Space

The Blue entity agent depends on a Situation-Awareness Model as defined by Endsley [23] for interacting with its operating environment. In this problem domain, the situation-awareness model is designed to have 29 multi-valued attributes which are based on the information of the *enemy* and the *terrain* for around 3.2876×10^4 possible situations.

Perception Layer: Information about the environment gathered using the primary sensory apparatus is represented at the perception layer. There are nine values each for *Enemy-Direction* and *SafeArea-Direction* attributes, eight values for the *Enemy-Orientation* attribute and two values for each of the *Adjacent-Location* attribute in each compass direction. Together, there is a total of 90 possible combinations from 11 types of attributes.

Comprehension Layer: Information at this layer is derived using information from the perception layer using hard-coded domain-specific knowledge. There are ten attributes for the comprehension layer. Among the ten attributes, there are three values for *Enemy-Location* attribute, seven values for *Enemy-Proximity* attribute and five values for the *Traversability* attribute in each compass direction. These three types of attributes gives a total of 3.2778×10^4 possible combinations at the comprehension layer.

Projection Layer: The projection of the situation is derived using information from the perception and comprehension layer. In this work, there is only one projection attribute on the presence of threat using the orientation and position of the adversary. This projection of threat is evaluated for each of the eight compass directions. For a single adversary, this give a total of 8 combinations for this *projection of threat*.

B. The Action Space

The Blue entity agent evades the pursuit of the Red entity agent by moving in a particular compass direction. Therefore, the action space is comprised of the eight compass directions - north, northeast, east, southeast, south, southwest, west and northwest - as the consequent of the decision-making task. The effect of the evade directions to the situation is learned and may be exploited for subsequent decision-making instances.

C. The Reward Space

The reward attributes refer to the sensory information used to quantify the effect of an action choice on a situation. Trends of these reward attributes are used to derive the immediate reward factor of the action choice. The reward attributes specific to this problem domain are presented in Table I.

TABLE I
Reward Attributes for PE problem domain

Reward Attribute	Positive	Negative
Proximity with Adversary	Increase in proximity with adversary	Decrease in proximity with adversary
Orientation w.r.t. Direction of Adversary	Facing in the direction of the adversary	Facing away from the direction of the adversary
Spaciousness of resultant Destination	Move into a more spacious location	Move into a more constrained location
Proximity to Safe Area	Move closer to a safe area	Move further away from a safe area
Presence of Obstacle	Move to an obstacle-free location	Move to a blocked location
Attacked by Adversary	Not attacked by adversary	Is attacked by adversary

VII. Experimental Results

Experiments are conducted using the complex PE problem to evaluate the efficacy of the proposed Knowledge-based Exploration Strategy. For the PE problem domain, TD-FALCON coupled with the relevant exploration strategy is tasked to learn the effective evasive maneuvers efficiently. The main performance indicators are the mission completion rate Ω_{mc} and the code population Ω_{cp}.

Mission Completion Ω_{mc}: 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 its adversary. No time-out is included in this PE problem domain. For each training iteration i, the Blue entity agent fails to complete the search mission when it is eliminated by the Red entity agent. In this sense, Ω_{mc} is the percentage of the number of times the Blue entity agent completes the search mission κ over ι training iterations, implying $\kappa \leq \iota$.

Code Population Ω_{cp}: The number of cognitive nodes at each training iteration i is termed as the Code Population Ω_{cp}. TD-FALCON always has an uncommitted cognitive node to learn an adequately distinct state-action pair. Without pruning, the growth of the code population is proportionally correlated to the rate of exploration. It is expected to saturate when the effective evasive strategies are learned.

Five different exploration strategies are used with TD-FALCON in three different configurations. TD-FALCON is used with and without *prior* knowledge and pruning to give three different configurations. Each experiment is conducted for 2000 training iterations. Each set of experimental result is averaged using 20 runs of the same experiment. In addition, for trending purpose only and to minimize non-essential fluctuations, every 100 data points are averaged to give 20 representative data points. The controlled parameters used for the experiments are presented in Table II.

TABLE II
Parameters of TD-FALCON and Action Selection Policy

FALCON Parameters for $k = \{1, 2, 3\}$		**TD Learning Parameters**	
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	Initial Q-Value	0.5
Learn Vigilance ρ_l^{ck}	0.85,1.0,0.45	Discount Factor γ	0.1
Perform Vigilance ρ_p^{ck}	0.85,0.0,0.45		
Contribution Parameters γ^{ck}	0.5,0.5,0.0		
ϵ-Greedy Policy Parameters			
Initial ϵ Value	0.8	ϵ Decay Rate	0.0005

The desirability discriminant parameters μ^+ and μ^- are set to 0.55 and 0.45 respectively. The temperature T for Simulated Annealing-based Exploration Strategy is set to 1.0 while the cooling factor α is set to 0.9. For the Tabu Search-based Exploration strategy, the duration at which an action choice remains tabu is set at 10 exploration cycles. This tabu duration is reduced at a step size of 1. The Neighborhood Search-based Exploration Strategy has no external parameter. The ϵ-greedy method is used all configurations.

IF	EnemyDirection	=	South
THEN	EvadeDirection	=	North
Statistics	Reward	=	0.75
	Confidence	=	1.0

Fig. 3. A trivial sample of an *inserted* Propositional Rule. Reward indicates effectiveness of the rule while Confidence indicates the usefulness of the rule.

A. Effect of TD-FALCON on Exploration Strategies

TD-FALCON without *prior* knowledge and pruning is used in this set of experiments. It uses five different exploration strategies for learning evasive strategies in the PE problem domain. The learning efficiency due to the proposed Knowledge-based exploration strategy is benchmarked against four other exploration strategies in this experiment.

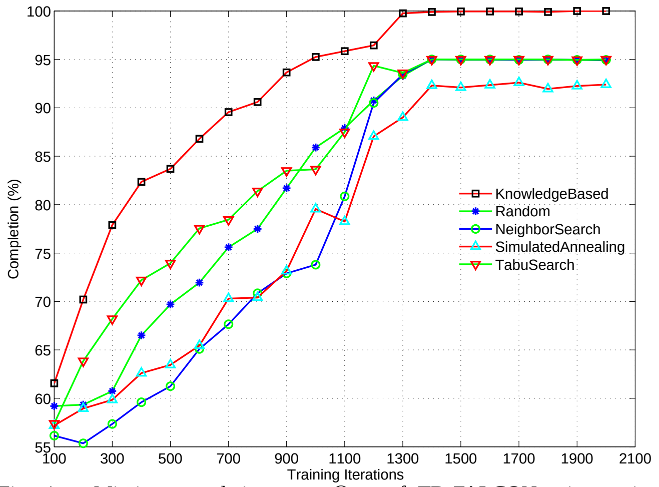

Fig. 4. Mission completion rates Ω_{mc} of TD-FALCON using various exploration strategies.

As shown in Fig. 4, the KnowledgeBased configuration is shown to have the best Ω_{mc} among the exploration strategies throughout the training process. It is also the only exploration strategy to saturate to 100% Ω_{mc}. Random, TabuSearch and NeighborSearch configurations saturates at 95% Ω_{mc} while SimulatedAnnealing fluctuates at about 92% Ω_{mc}.

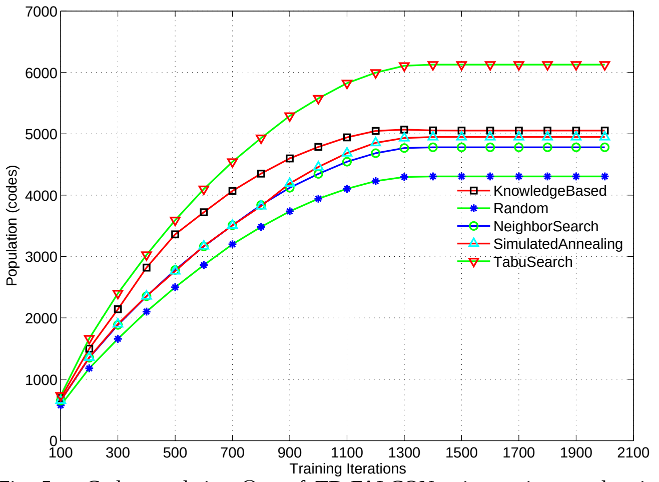

Fig. 5. Code population Ω_{cp} of TD-FALCON using various exploration strategies.

Plots of Ω_{cp} in Fig. 5 show KnowledgeBased configuration having similar Ω_{cp} with SimulatedAnnealing and NeighborSearch configurations while attaining higher Ω_{mc} than these two configurations. The TabuSearch configuration has the highest Ω_{cp} but the second lowest Ω_{mc} among the five configurations. Such an observation highlights the effectiveness of the proposed Knowledge-based Exploration Strategy in learning more effective knowledge. This is because the negative chunks are used to prevent further exploration of actions known to be ineffective. This has helped KnowledgeBased to attain higher Ω_{mc} than all other exploration strategies.

B. *Effect of* Prior *Knowledge on Exploration Strategies*

Further experiments are conducted using TD-FALCON (denoted using PriorKB) inserted with *prior* knowledge using the technique described in Section IV-A. The effect of inserting *prior* knowledge such as the one illustrated in Fig. 3 into TD-FALCON on the exploration strategies is studied here. The same five exploration strategies are used with PriorKB version of TD-FALCON.

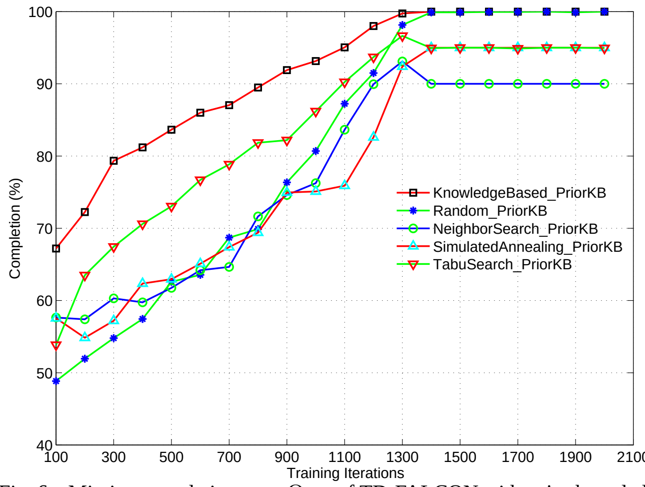

Fig. 6. Mission completion rates Ω_{mc} of TD-FALCON with *prior* knowledge using various exploration strategies.

From Fig. 6, the KnowledgeBased_PriorKB configuration already has close to 70% Ω_{mc} in the first 100 training iterations. It stays above all the other configurations right up to its 100% Ω_{mc} saturation at around the 1400^{th} training iteration. Beginning at the lowest Ω_{mc} level, the Random_PriorKB configuration is the only other configuration to attain 100% Ω_{mc}. TabuSearch_PriorKB and SimulatedAnnealing_PriorKB saturate to the next highest Ω_{mc} level while NeighborSearch_PriorKB converges to the lowest Ω_{mc} level.

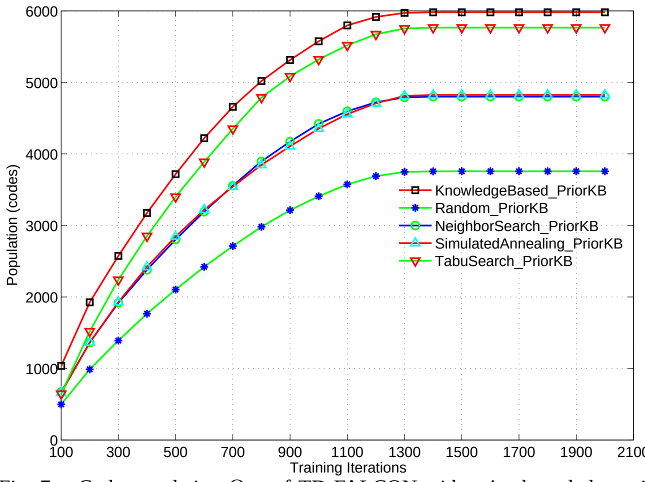

Fig. 7. Code population Ω_{cp} of TD-FALCON with *prior* knowledge using various exploration strategies.

From Fig. 7, KnowledgeBased_PriorKB is observed with the highest Ω_{cp} among the five configurations. This is followed closely by TabuSearch_PriorKB. SimulatedAnnealing_PriorKB and NeighborSearch_PriorKB saturate to similar Ω_{cp} while Random_PriorKB has quite expectedly saturate to the lowest Ω_{cp}. Therefore, it is clear from Fig. 6 and Fig. 7 that inserting *prior* knowledge has a positive impact on the proposed Knowledge-based Exploration Strategy. Combining the use of *prior* knowledge and learned knowledge to direct exploration has resulted in the discovery of more effective knowledge. Unlike the Knowledge-based Exploration Strategy, the other exploration strategies are unable to benefit as much from the *prior* knowledge because they do not make use of the learned and inserted knowledge during exploration.

C. *Effect of Pruning on Exploration Strategies*

This round of experiments are conducted using TD-FALCON (denoted using Prune) with the pruning ability described in Section IV-B. The effect of pruning away ineffective knowledge chunks on the exploration strategies is studied here. In the experiments, cognitive nodes with confidence level below 35% of the average confidence level are pruned. The same five exploration strategies are used with the Prune version of TD-FALCON.

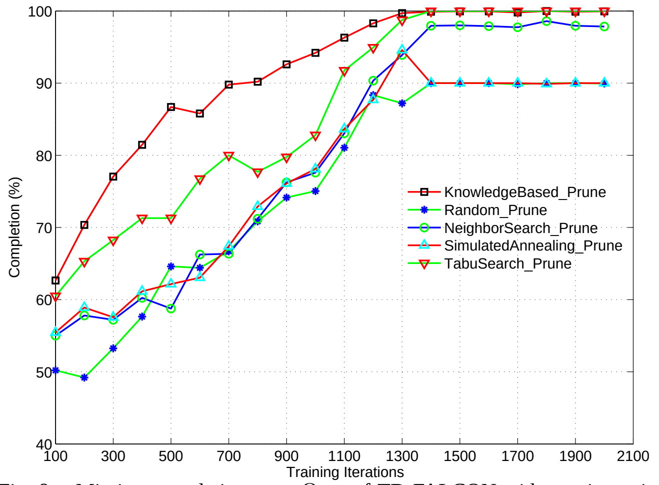

Fig. 8. Mission completion rates Ω_{mc} of TD-FALCON with pruning using various various exploration strategies.

With reference to Fig. 8, the KnowledgeBased_Prune configuration has the most efficient Ω_{mc} profile than the other configurations. Though TabuSearch_Prune has also converged to 100% Ω_{mc}, it has larger fluctuations prior to saturation. All the other exploration strategies have quite similar profile during training. On saturation, NeighborSearch_Prune has the next highest Ω_{mc} while Random_Prune and SimulatedAnnealing_Prune saturate to the same Ω_{mc} level.

From Fig. 9, due to the effect of pruning, there is a downward shift of the saturation levels of Ω_{cp} for all five configurations when compared with the saturation levels of Ω_{cp} seen in Fig. 5 and Fig. 7. However, the correlations of Ω_{cp} among the configurations remains unchanged from what are observed in Fig. 7. From the experimental results, the

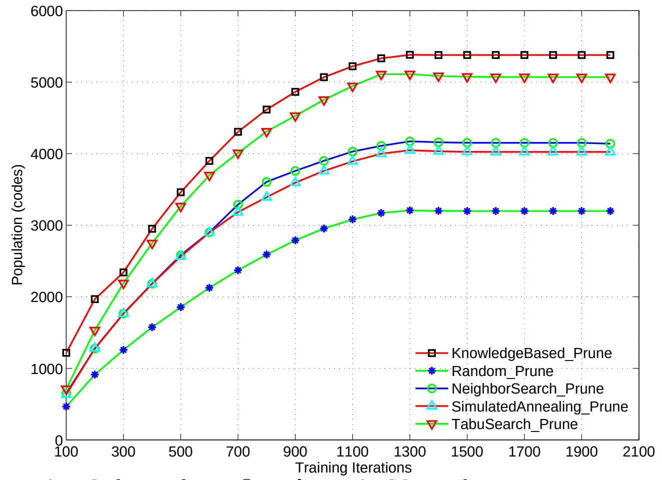

Fig. 9. Code population Ω_{cp} of TD-FALCON with pruning using various exploration strategies.

proposed Knowledge-based Exploration Strategy remains as robust as it has in the earlier two experiments. On the other hand, there appears to be some significant impact on how the other exploration strategies performs.

VIII. Conclusion

This paper has proposed the Knowledge-based Exploration strategy, which leverages the learned knowledge of a learning agent to direct exploration during RL. Specifically, the strategy searches for knowledge chunks in the learning agent encoding the outcome of a given action in similar situations and filters away those actions known to lead to negative outcomes from the list of possible actions for exploration. This has led to a more targeted and efficient exploration of the action space.

Using several experiments based on a complex PE problem, the proposed Knowledge-Based Exploration strategy was compared to the baseline random exploration and three other directed exploration strategies, namely the Neighborhood Search-based Exploration Strategy, the Simulated Annealing-based Exploration Strategy and the Tabu Search-based Strategy, in terms of success rates (mission completion) and model complexity (code population). Our experiments show the Knowledge-based Exploration Strategy to be robust and consistent when used with TD-FALCON without and with the use of knowledge insertion and pruning. Specifically, the Knowledge-based Exploration Strategy converged to 100% mission completion rates for all configurations of TD-FALCON. In contrast, none of the configurations of the same TD-FALCON using all the other exploration strategies saturate to 100% mission completion rates.

The Knowledge-based Exploration Strategy retains knowledge of the ineffective action choices to direct exploration towards the discovery of the effective action choices. The impact of such an approach is in the higher code population as compared to the other directed exploration strategies. The Tabu Search-based Exploration Strategy, being the most aggressive of the alternative exploration strategies, has the next highest code population. On the other hand, the least aggressive Random Exploration Strategy has, expectedly, the lowest code population. In view of the current limitation, our subsequent work may focus on better management of the cognitive node population by not exploring unnecessarily while maintaining the performance level. The duration of the training process may also need to be reduced significantly by having a faster saturation rate for a quicker turnaround time.

Acknowledgment

This work is supported in part by the DSO National Laboratories under research grant DSOCL11258.

References

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

[2] S. B. Thrun, “Efficient exploration in reinforcement learning,” Carnegie Mellon University, School of Computer Science, Tech. Rep., 1992.

[3] K. Främling, “Guiding exploration by pre-existing knowledge without modifying reward,” *Neural Networks*, vol. 20, pp. 736–747, 2007.

[4] F. Fernández and M. Veloso, “Probabilistic policy reuse in a reinforcement learning agent,” in *Proceedings of the International Conference on Autonomous Agents*, 2006, pp. 720–727.

[5] N. A. Vien, N. H. Viet, S.-G. Lee, and T.-C. Chung, “Heuristic search based exploration in reinforcement learning,” *Computational and Ambient Intelligence*, vol. 4507, pp. 110–118, September 2007.

[6] J. Parviainen and M. Eriksson, “Negative knowledge, expertise and organisations,” *Int. J. Management Concepts and Philosophy*, vol. 2, no. 2, pp. 140–153, 2006.

[7] 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 Transactions on Neural Networks*, vol. 19, no. 2, pp. 230–244, February 2008.

[8] A.-H. Tan, G. A. Carpenter, and S. Grossberg, “Intelligence through interaction: Towards a unified theory for learning,” in *Proceedings of the 4^{th} International Symposium on Neural Networks: Advances in Neural Networks*, 2007, pp. 1094–1107.

[9] C. J. C. H. Watkins and P. Dayan, “Q-Learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

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

[11] L. D. Berkovitz, *Advances in game theory*. Princeton University Press, 1971, ch. A Variational Approach to Differential Games.

[12] T.-H. Teng and A.-H. Tan, “Cognitive agents integrating rules and reinforcement learning for context-aware decision support,” in *Proceedings of the IAT*, December 2008, pp. 318–321.

[13] T.-H. Teng, A.-H. Tan, Y.-S. Tan, and A. Yeo, “Self-organizing Neural Networks for Learning Air Combat Maneuvers,” in *Proceedings of the IJCNN*, June 2012, pp. 2859–2866.

[14] T.-H. Teng, A.-H. Tan, W.-S. Ong, and K.-L. Lip, “Adaptive CGF for Pilots training in Air Combat Simulation,” in *Proceedings of the 15^{th} International Conference on Information Fusion*, July 2012, pp. 2263–2270.

[15] A.-H. Tan, “FALCON: A Fusion Architecture for Learning, Cognition, and Navigation,” in *Proceedings of the IJCNN*, 2004, pp. 3297–3302.

[16] ——, “Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning,” in *Proceedings of the IJCAI*, January 2007, pp. 1071–1076.

[17] A. Pérez-Uribe, “Structure-adaptable digital neural networks,” Ph.D. dissertation, Swiss Federal Institute of Technology-Lausanne, 2002.

[18] T.-H. Teng, Z.-M. Tan, and A.-H. Tan, “Self-organizing neural models integrating rules and reinforcement learning,” in *Proceedings of the IJCNN*, June 2008, pp. 3770–3777.

[19] E. Y. Rodin, “A Pursuit-Evasion Bibliography,” *Computers & Mathematics with Applications*, vol. 13, no. 1-3, pp. 275–340, 1987.

[20] A. Dumitrescu, H. Kok, I. Suzuki, and P. Żyliński, “Vision-based pursuit-evasion in a grid,” *SIAM Journal on Discrete Mathematics*, vol. 24, no. 3, pp. 1177–1204, 2010.

[21] S. Ficici and J. Pollack, “Statistical reasoning strategies in the pursuit and evasion domain,” in *Advances in Artificial Life*, ser. Lecture Notes in Computer Science, D. Floreano, J.-D. Nicoud, and F. Mondada, Eds. Springer Berlin / Heidelberg, 1999, vol. 1674, pp. 79–88.

[22] M. E. Harmon, L. C. Baird, and A. H. Klopf, “Advantage updating applied to a differential game,” *Advances in Neural Information Processing Systems*, vol. 7, pp. 353–360, 1995.

[23] M. R. Endsley, “Designing for situation awareness in complex systems,” in *Proceedings of the 2^{nd} International Workshop on Symbiosis of Humans, Artefacts and Environment*, 2001, pp. 1–14.