1-2012

Self-regulating action exploration in reinforcement learning

Teck-Hou TENG

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

Yuan-Sin TAN

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 13 (2012) 18 – 30

Procedia Computer Science

Proceedings of the International Neural Network Society Winter Conference (INNS-WC 2012)

Self-Regulating Action Exploration in Reinforcement Learning

Teck-Hou Teng[a,*], Ah-Hwee Tan[a], Yuan-Sin Tan[b]

[a]*School of Computer Engineering, Nanyang Technological University*
[b]*DSO National Laboratories*

Abstract

The basic tenet of a learning process is for an agent to learn for only as much and as long as it is necessary. With reinforcement learning, the learning process is divided between exploration and exploitation. Given the complexity of the problem domain and the randomness of the learning process, the exact duration of the reinforcement learning process can never be known with certainty. Using an inaccurate number of training iterations leads either to the non-convergence or the over-training of the learning agent. This work addresses such issues by proposing a technique to self-regulate the exploration rate and training duration leading to convergence efficiently. The idea originates from an intuitive understanding that exploration is only necessary when the success rate is low. This means the rate of exploration should be conducted in inverse proportion to the rate of success. In addition, the change in exploration-exploitation rates alters the duration of the learning process. Using this approach, the duration of the learning process becomes adaptive to the updated status of the learning process. Experimental results from the *K*-Armed Bandit and Air Combat Maneuver scenario prove that optimal action policies can be discovered using the right amount of training iterations. In essence, the proposed method eliminates the guesswork on the amount of exploration needed during reinforcement learning.

Keywords: reinforcement learning, exploration-exploitation dilemma, *k*-armed bandit, pursuit-evasion, self-organizing neural network

1. Introduction

The most desirable form of learning is to spend just the right amount of time and effort to learn the knowledge necessary for a task. With reinforcement learning (RL), the learning process switches between exploration and exploitation [1]. The need to discover action policies more optimal than the existing ones is met by exploring the action space. The effectiveness of the learned action policies are probed as they are exploited during learning. Consequentially, it must be able to eventually settle into the full exploitation of the learned action policies.

Exploration and exploitation are known to be balanced using a variety of solutions [2, 3, 4, 5]. Greedy strategies such as the ϵ-greedy method and prioritized sweeping, randomized strategy such as the Boltzmann exploration

*Corresponding author
Email address: thteng@ntu.edu.sg (Teck-Hou Teng)
URL: www.ntu.edu.sg/home/thteng (Teck-Hou Teng)

1877-0509 © 2012 Published by Elsevier B.V. Selection and/or peer-review under responsibility of Program Committee of INNS-WC 2012
Open access under CC BY-NC-ND license. doi:10.1016/j.procs.2012.09.110

and interval-based techniques such as the interval estimation algorithm are used to balance exploitation and exploration [6]. Despite all these works, it remains a challenge to explore for just the right amount of time. In addition, there seems to be a lack of an adaptive approach to ensure the right amount of training iteration for the RL process.

In this respect, this work proposes a solution to self-regulate action exploration during RL. The proposed solution addresses the issues of uncertainty over the amount of exploration and the length of the training process. The aim is to discover the optimal action policies using just the right amount of training iterations. During learning, at a fixed interval known as a window, the value of ϵ of the ϵ-greedy method is revised using the interval success rates. The exploration of the action space and training duration are then self-regulated using the revised ϵ. In effect, this proposed technique has adapted two external parameters - exploration rates and training duration - using a learning status parameter known as the interval success rate.

In this work, learning and exploitation of the action policies are carried out using an ART-based Neural Network known as the Fusion Architecture for Learning and Cognition (FALCON) [7]. It is known to be capable of incremental learning in real time for a variety of learning tasks [8, 9, 10]. It is used within the RL framework where the proposed solution of self-regulating action exploration (SRE) is applied. Empirical results collected from experiments conducted using two multi-state problem domains (the Air Combat Maneuver scenario and K-Armed Bandit problem) have demonstrated the ability to correlate the exploration rate to the updated status of the learning process and the length of the training process is regulated in real time during RL. Therefore, using the proposed solution ensures the right amount of exploration and the training iterations are allocated to the RL process.

The rest of the paper is organized as follows. Survey of the related works is presented in Section 2. This is followed by the summarized presentation of FALCON using temporal difference method during RL in Section 3. The proposed technique to self-regulate action exploration is detailed in Section 4. Introduction to the problem domains, description of respective experiments as well as analysis of results are provided in Section 5. Last but not least, the conclusions as well as future directions for this work are provided in Section 6.

2. Related Work

Reinforcement learning (RL) uses exploration to discover new action policies and exploitation to apply the learned action policies to the situations [1]. The K-Armed Bandit problem is widely used to study the exploration-exploitation dilemma [11]. In a multi-state environment [12], the task of discovering the optimal action policies for the states is a non-trivial issue that has attracted wide attention [13, 4, 5, 14].

The two facets of the exploration-exploitation dilemma - large state-action space and non-stationary environment - are addressed by combining recency-based exploration (RBE) with a detect-and-explore (DAE) algorithm [3]. The Boltzmann action selection policy is used to decide between exploitation and exploration. However, from their experiments conducted using two non-stationary navigation-based scenarios, their RB-DAE algorithm is only capable of performance level between the RBE and DAE algorithms. On the other hand, there is a work based on the bandit problem theory to derive exploration bonus to address the problem of uncertainty in exploring the states [12]. Extensive experimental results are presented to show greater learning efficiency over the conventional approaches.

The Explicit, Explore or Exploit (E^3) algorithm was proposed to identify the optimal policy using a *balanced wandering* phase where the least used action choices are explored on entering into a particular state [5]. The *balanced wandering* approach is replaced with an adaptive exploration phase in [14]. However, both algorithms are only presented analytically. The empirical performance of these two approaches for standard RL problems remains unclear.

Like the Boltzmann distribution, the ϵ-greedy method is commonly used to balance between exploration and exploitation. The conventional ϵ-greedy method decays ϵ linearly to gradually shift from the exploration to the exploitation of the learned action policies. In general, high exploration, i.e. high ϵ, is preferred at the beginning to spur the discovery of effective action policies [4]. Alternatively, ϵ may be moderated using time-based discounts of the past rewards [13]. Another method is to use value function error to control the value of ϵ [2]. Experimental results only show it to be more robust but not necessarily much better over the conventional action selection policies such as the ϵ-greedy method and the softmax method.

The surveyed works in [3, 12, 5, 14] improves learning efficiency using different exploration-specific strategies while a number of surveyed works such as [4, 13, 2] modifies ϵ for the similar effect. However, not all of them are able to clearly illustrate their effectiveness using experimental results. Also, experiment results for some other works are rather inconclusive in some sense. Like [4, 13, 2], this work controls the value of ϵ to self-regulate exploration rate and training duration. However, the self-regulation of training duration using ϵ remains a novel concept. Also, the multi-state K-armed bandits problem domain used in this work marks the attempted bandits to prevent the "*sticking* problem" experienced in [12]. In addition, using a commercial-grade simulation platform, the proposed SRE algorithm used with FALCON is also illustrated using a pursuit-evasion problem in three-dimensional airspace.

3. The Reinforcement Learning Model

In this work, the learning agent is driven by a self-organizing neural network known as FALCON [7]. Based on the adaptive resonance theory (ART), it can learn incrementally and generalize on the vector patterns. Using reinforcement learning, action policies are discovered during real-time interactions with the environment.

3.1. FALCON Model and Processes

The FALCON network [7] employs a 3-channel architecture (Fig. 1), comprising a category field F_2^c and three input fields, namely a sensory field F_1^{c1} for representing current states, an action field F_1^{c2} for representing actions, and a reward field F_1^{c3} for representing reinforcement values. A brief summary of the FALCON generic network dynamics, based on fuzzy ART operations [15], is described below.

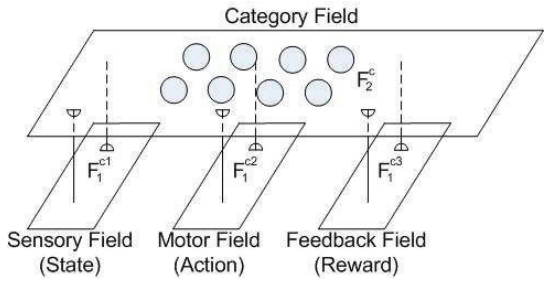

Fig. 1. An illustration of the FALCON Architecture.

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

Activity vectors: Let $\mathbf{x}^{ck}$ denote the F_1^{ck} activity vector for $k = 1, \ldots, 3$. Let $\mathbf{y}^c$ denote the F_2^c activity vector. Upon input presentation, $\mathbf{x}^{c1} = \mathbf{S}$, $\mathbf{x}^{c2} = \mathbf{A}$, and $\mathbf{x}^{c3} = \mathbf{R}$.

Weight vectors: Let $\mathbf{w}_j^{ck}$ denote the weight vector associated with the jth node in F_2^c for learning the input patterns in F_1^{ck} for $k = 1, \ldots, 3$. Initially, F_2^c contains only one *uncommitted* node and its weight vectors contain all 1's. When an *uncommitted* node is selected to learn an association, it becomes *committed*.

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

Code activation: A bottom-up propagation process first takes place in which the activities (known as choice function values) of the cognitive nodes in the F_2^c field are computed. Specifically, given the activity vectors $\mathbf{x}^{c1}$, $\mathbf{x}^{c2}$ and $\mathbf{x}^{c3}$ (in the input fields F_1^{c1}, F_1^{c2} and F_1^{c3} respectively), for each F_2^c node j, the choice function T_j^c is computed as follows:

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

where the fuzzy AND operation $\wedge$ is defined by $(\mathbf{p} \wedge \mathbf{q})_i \equiv min(p_i, q_i)$, and the norm $|.|$ is defined by $|\mathbf{p}| \equiv \sum_i p_i$ for vectors $\mathbf{p}$ and $\mathbf{q}$. In essence, the choice function T_j computes the similarity of the activity vectors with their respective weight vectors of the F_2^c node j with respect to the norm of the weight vectors.

Code competition: A code competition process follows under which the F_2^c node with the highest choice function value is identified. The winner is indexed at J where

$$J = \arg\max_j \{T_j^c : \text{for all } F_2^c \text{ node } j\}$$

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

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

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

The match function computes the similarity of the activity and weight vectors with respect to the norm of the activity vectors. Together, the choice and match functions work cooperatively to achieve stable coding and maximize code compression.

When resonance occurs, learning ensues. If any of the vigilance constraints is violated, mismatch reset occurs in which the value of the choice function T_J^c is set to 0 for the duration of the input presentation. The search process then selects another F_2^c node J under the revised vigilance criterion until a resonance is achieved. This search and test process is guaranteed to end as FALCON will either find a *committed* node that satisfies the vigilance criterion or activate an *uncommitted* node which would definitely satisfy the vigilance criterion due to its initial weight values of $1s$.

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

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

For an uncommitted node J, the learning rates β^{ck} are typically set to 1. For committed nodes, β^{ck} can remain as 1 for fast learning or below 1 for slow learning in a noisy environment. When an uncommitted node is selected for learning, it becomes *committed* and a new uncommitted node is added to the F_2^c category field.

3.2. Incorporating Temporal Difference Method

TD-FALCON [16] incorporates Temporal Difference (TD) methods to estimate and learn value functions of state-action pairs $Q(s, a)$ that indicates the goodness for taking a certain action a in a given state s. This is learned as the feedback signal and is used in the selection of the action choices.

As shown in Algorithm 1, given the current state s, TD-FALCON first decides between exploration and exploitation by following an action selection policy. For exploration, a random action is picked. For exploitation, TD-FALCON searches for optimal action through a direct code access procedure [17]. Upon receiving a feedback from the environment after performing the action, a TD formula is used to compute a new estimate of the Q-value for performing the chosen action in the current state. The new Q-value is then used as the teaching signal to TD-FALCON to learn the association of the current state and the chosen action to the estimated Q-value.

Iterative Value Estimation: A value function based on a temporal difference method known as Bounded Q-Learning [18] 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 $(1 - Q_j(s, a))$ 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 which is derived using

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

Algorithm 1 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* 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 state S, action A and reward R vectors for **Learning**
13: Update the current state $s = s'$
14: Repeat from Step 2 until s is a terminal state

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.

3.3. Pruning

Newly discovered action policies is learned as cognitive nodes during RL. However, quite a number of these cognitive nodes will become irrelevant as learning progresses. 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 [7] 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 is 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.

4. Self-Regulating Action Exploration

A method to self-regulate the action exploration and training duration is proposed in this section. This proposed method addresses issues pertaining to the lack of correlation with the actual status of the learning process when trying to balance between exploration and exploitation and the stochastic nature of the learning process. Details on the main features of the proposed method are provided in the subsequent sections.

4.1. The Action Selection Policy

As with many reinforcement learning solutions [7, 19], the ϵ-greedy action selection policy is used to balance between exploration and exploitation. With such an action selection policy, exploration is occurring at a *probability* of ϵ where $\epsilon \in [0, 1]$. The conventional approach is to use high ϵ for higher exploration rate at the onset of the learning process. At each training iteration, ϵ is linearly decayed using

$$\epsilon^{new} = \epsilon^{old} - \theta \tag{1}$$

where the decay rate θ is derived at the onset of the learning process, i.e. $n = 0$, using

$$\theta = \frac{\epsilon_0}{\mathcal{N}_\delta \mathcal{N}_0} \tag{2}$$

where ϵ_0 is the initial value of ϵ, $\mathcal{N}_\delta$ is the training proportion such that $(\mathcal{N} - \mathcal{N}_\delta \mathcal{N})$ is the number of training iterations where the learning agent operates in full exploitation mode and $\mathcal{N}_0$ is the initial number of training iterations. In this work, the total number of training iterations $\mathcal{N}$ is updated using

$$\mathcal{N} = \mathcal{N}_e + \mathcal{N}_r \tag{3}$$

where $\mathcal{N}_e$ denotes the elapsed training iterations and $\mathcal{N}_r$ denotes the remaining training iterations. Using the conventional method, $\mathcal{N}_r$ is constant and the change in ϵ is de-correlated from the updated status of the learning process.

Given the stochastic nature of the learning process, there is a non-zero probability that the optimal action policies may yet to be discovered when ϵ is fully decayed. Since there can be no further exploration when ϵ is fully decayed, the learning process will saturate at a non-optimal level. Therefore, for greater learning efficiency and effectiveness, this work proposes an approach to correlate the modification of ϵ and $\mathcal{N}_r$ to the updated status of the learning process using the interval success rates λ.

4.2. The Interval Success Rates

Unlike the time-discounted past rewards [13], the interval success rate $\lambda \in [0.0, 1.0]$ is used as an updated measure of efficacy of the learned action policies. It is based only on two types of outcome status - positive or negative - at the terminal states. Depending on the nature of the problem, a neutral outcome status may be perceived as a negative or positive outcome status. Regardless of the treatment of the neutral outcome status, the types of outcome status at the terminal states are limited to just the positive and negative.

Let $\mathbf{P}$ denotes the set of terminal states s_t whose outcome status O_{s_t} is positive, i.e. $O_{s_t} \in \mathbf{P}$ and let $\mathbf{N}$ denotes the set of terminal states s_t whose outcome status O_{s_t} is negative, i.e. $O_{s_t} \in \mathbf{N}$. The outcome at terminal state s_t is a quantitative measure η_{s_t} and the function $f(\eta_{s_t})$ is used to qualify this outcome status O_{s_t} as either positive or negative. The number of positive outcome status is represented using $\mathcal{N}_p$.

The number of positive outcome status $\mathcal{N}_p$ is gathered over a fixed number of training iterations $\mathcal{N}_w$ collectively known as a *window* such that $\mathcal{N}_p \leq \mathcal{N}_w$ where $\mathcal{N}_w \in [\varepsilon|\mathbf{A}|, \mathcal{N}]$, $\varepsilon \in (0.0, 1.0]$ and $|\mathbf{A}|$ denotes the size of action space $\mathbf{A}$. Using $\mathcal{N}_w \equiv \mathcal{N}$ reverts to a windowless learning process.

A two-mode self-regulating process is implemented using a *window*. At training iteration n, the self-regulating process is in the *window-open* mode when (n mod $\mathcal{N}_w \neq 0.0$.) and is in the *window-close* mode when (n mod $\mathcal{N}_w \equiv 0.0$). The number of positive outcome status $\mathcal{N}_p$ is gathered during the *window-open* mode. At the *window-close* mode, the interval success rate λ is derived using

$$\lambda = \frac{\mathcal{N}_p}{\mathcal{N}_w} \tag{4}$$

The number of positive outcome status $\mathcal{N}_p$ is also reset at the *window-close* mode. This approach reduces the influence of the outcome status at the terminal states to the size of the *window*. This is found to have a stabilizing effect on the learning process.

4.3. Regulating the Exploration Rate

It can be inferred from Section 4.1 that the exploration rate can be controlled using ϵ. The windowing concept introduced in Section 4.2 is also used for the modification of ϵ. At the *window-close* mode, i.e. (n mod $\mathcal{N}_w$) $\equiv 0$, ϵ is updated using

$$\epsilon^{\text{new}} = f(1-\lambda)\left\{\kappa(1-\lambda) + (1-\kappa)\epsilon^{\text{old}}\right\} \tag{5}$$

where $\kappa \in [0.0, 1.0]$ is the ϵ-adaptation rate and $f(x)$ is a step function such that

$$f(x) = \begin{cases} 1 & \text{when } x > 0 \\ 0 & \text{when } x \leq 0 \end{cases}$$

This means $\epsilon^{new} > 0.0$ when $0.0 \leq \lambda < 1.0$. Using (5) sets the exploration rate to be inversely proportional to the interval success rate λ. Using this approach places the right amount of emphasis on the exploration of the action space $\mathcal{A}$.

At the *window-open* mode, the shift of the balance between exploration and exploitation is linearly moderated using the linear decay method in (1) using the same ϵ-decay rate θ derived from the onset using (2). This gradual shift from exploration to exploitation allows for more exploitation of the discovered action policies over time.

4.4. Regulating the Training Duration

From the onset of the learning process, the ϵ-decay rate is initialized using (2). Thereafter, it remains unchanged during learning. Given that ϵ is modified using (5) at the *window-close* mode, the *remaining* number of training iterations $\mathcal{N}_r$ is derived using

$$\mathcal{N}_r = \frac{\epsilon^{new}}{\mathcal{N}_\delta \theta} - \left(\frac{\epsilon^{new}}{\mathcal{N}_\delta \theta} \bmod \mathcal{N}_w \right) + \mathcal{N}_w \tag{6}$$

From (6), it can also be seen that the training duration $\mathcal{N}_r$ becomes $\mathcal{N}_w$ when $\epsilon = 0.0$. Given that (6) is only used at the *window-close* mode, $\mathcal{N}_e$ is always ($\mathcal{N}_e \bmod \mathcal{N}_w \equiv 0$). Using (6) to derive $\mathcal{N}_r$ ensures the total number of training iteration $\mathcal{N}$ updated using (3) remains as the multiple of window size $\mathcal{N}_w$. Convergence is obtained when the interval success rate $\lambda = 1.0$ at the last *window-close* mode or when ϵ is fully decayed within the last *window-open* mode.

Algorithm 2 Self-Regulating Action Exploration (SRE)

```
Initialize ε_0, N_0, N_δ
Initialize θ using (2)
Set N = N_0
for n = 0 to N do
    if (n mod N_w) ≠ 0 then
        Update ε using (1) {See Section 4.1}
        Tracks N_p
    else if (n mod N_w) ≡ 0 then
        Derive λ using (4) {See Section 4.2}
        Update ε using (5) {See Section 4.3}
        Derive N_r using (6) {See Section 4.4}
        Update N using (3)
        Reset N_p
    end if
end for
```

Self-regulating the exploration rate and the training duration allows the ϵ-greedy method to be adaptive towards the status of the learning process and eliminates the need to estimate the number of training iterations required to ensure convergence. An outline of the novel method to self-regulate the action exploration is presented in Algorithm 2.

5. Experiments

Two multi-state markov decision process (MDP) problem domains are used to evaluate the proposed technique of self-regulating action exploration during reinforcement learning. Using a commercial-grade simulation platform, the first problem domain models a 1-v-1 air combat maneuvering (ACM) scenario between two Computer-Generated Forces (CGFs). The other problem domain is the K-Armed Bandit. It is widely used for illustrating the exploration-exploitation dilemma [1].

Table 1. Parameters of TD-FALCON

TD–FACLON Parameters	
Choice Parameters ($\alpha^{c1}, \alpha^{c2}, \alpha^{c3}$)	0.1,0.1,0.1
Learning Rates ($\beta^{c1}, \beta^{c2}, \beta^{c3}$)	1.0,1.0,1.0
Contribution Parameters ($\gamma^{c1}, \gamma^{c2}, \gamma^{c3}$)	0.33,0.33,0.33
Perform Vigilance ($\rho_p^{c1}, \rho_p^{c2}, \rho_p^{c3}$)	0.0,0.0,0.45
Learn Vigilance ($\rho_l^{c1}, \rho_l^{c2}, \rho_l^{c3}$)	0.95,1.0,0.45
Temporal Difference Learning Parameters	
Learning Rate α	0.5
Discount Factor γ	0.1
Initial Q-Value	0.5

Table 2. Parameters of the SRE algorithm and Pruning strategy

SRE Algorithm Parameters	
Initial ϵ Value	0.9
Training Proportion $\mathcal{N}_\delta$	0.95
ϵ-adaptation rate κ	1.0
Pruning Strategy Parameters	
Confidence decay rate ζ	0.003
Confidence reinforcement rate η	0.05
Old age σ^{old}	20

5.1. The Air Combat Maneuver scenario

The Air Combat Maneuvering (ACM) scenario is based on a classical 1-v-1 pursuit-evasion problem in three-dimensional airspace [20]. Like [10], the adaptive CGF is represented as the Blue CGF while the non-adaptive CGF is represented as the Red CGF. Both CGFs are tasked to out-maneuver each other to enter into a favorable position to eliminate each other using air-to-air missiles. Their state space is made up of 15 propositional symbols and their action space is made up of 13 air combat maneuvers.

In this experiment, only the Blue CGF adapts its air combat maneuvers using TD-FALCON based on either the proposed SRE algorithm or the conventional ϵ-greedy method with a linear decay schedule. The Red CGF does not learn and is only driven using the built-in doctrine of the simulation platform. The reinforcement learning problem here is for the Blue CGF to discover the most effective action policies for the different situations to eliminate the Red CGF in a consistent manner in 1-v-1 dogfights.

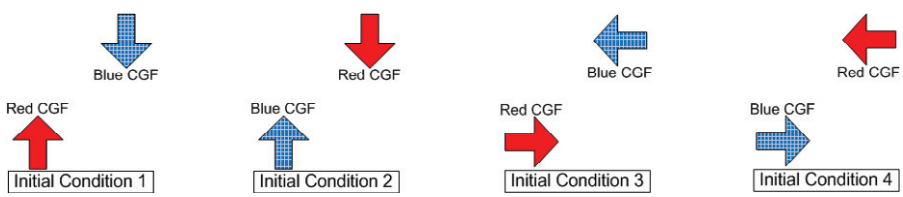

Fig. 2. Illustration of the four initial conditions used for the ACM experiments

Experiments based on the 1-v-1 ACM scenario are conducted to compare the performance of the adaptive Blue CGF driven by the proposed SRE algorithm (refer to as SRE20) and another adaptive Blue CGF driven by the conventional ϵ-greedy method with a linear decay schedule (refer to as Linear20). As illustrated in Fig. 2, four different initial conditions are rotated during reinforcement learning. For the ACM scenario, TD-FALCON and the SRE algorithm are configured using the set of parameters illustrated in Table 1 and Table 2 respectively.

The interval success rate λ for this ACM scenario, referred to as HasKill, is the number of eliminations of the Red CGF by the Blue CGF. Ten sets of results are averaged and condensed over the size of the window for

both configurations. The SRE-driven Blue CGF in this ACM scenario uses a *window* comprising of 5 training iterations. The initial training iterations $\mathcal{N}_0$ for both configurations is set at 20, i.e. $\mathcal{N}_0 = 20$.

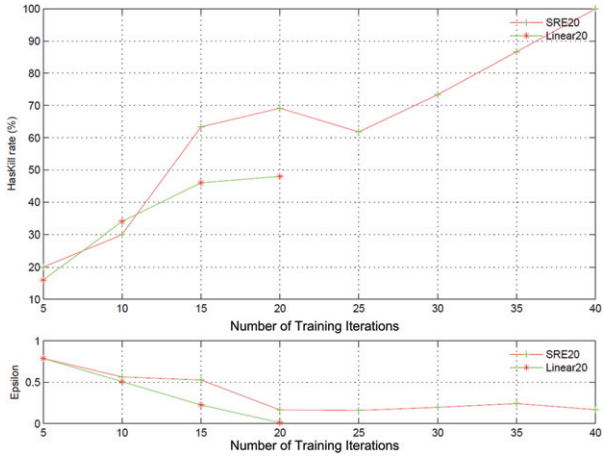

Fig. 3. Comparison of HasKill performance indicator for the ACM scenario using $\mathcal{N}_0 = 20$, $\epsilon_0 = 0.9$

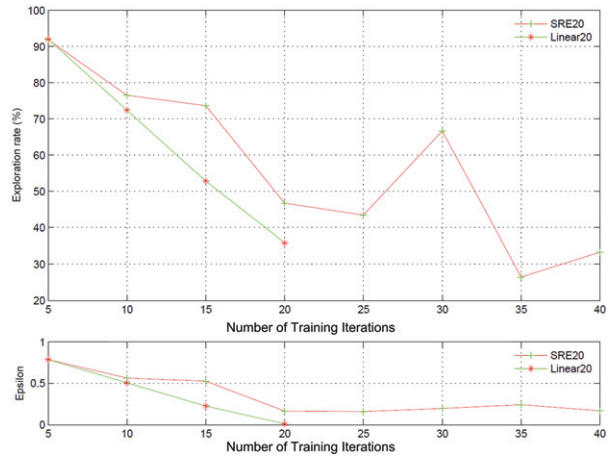

Fig. 4. Comparison of exploration rates for the ACM scenario using $\mathcal{N}_0 = 20$, $\epsilon = 0.9$

From Fig. 3, though both configurations are incapable of attaining 100% HasKill rate using $N_0 = 20$, SRE20 is observed with a higher HasKill rate than Linear20. From the bottom plot of Fig. 3, the SRE algorithm keeps ϵ little changed when the HasKill rate is found to be only at 30%. Higher ϵ leads to higher exploration rate for SRE20 which improves the chance of finding more optimal action policies than Linear20. As a reminder, the top plot of Fig. 3 compares the plots by aggregating 10 runs of the SRE20 and the Linear20 configurations. Due to the adaptation of the training duration to the status of the learning process, the total number of training iteration for each RL session for SRE20 is actually different. In fact, an average of 25.6 training iterations from ten sets of results for the SRE20 configuration is observed. The aggregated plot of SRE20 shows some runs of the experiment used up to another 20 training iterations to attain 100% HasKill rate. Therefore, it is confirmed using 20 training iterations is insufficient to attain convergence for the ACM scenario.

The impact of ϵ on the exploration rates is revealed using Fig. 4. Expectedly, the exploration rate for Linear20 drops linearly as ϵ is decayed linearly. It is also observed that the fluctuations of the exploration rate for SRE20 track the value of ϵ plotted at the bottom plot of Fig. 4. As a reminder, exploration is occurring at a probability of ϵ, not in direct proportion of ϵ. Therefore, some amount of de-correlation between the exploration rate and ϵ can be observed at around the 30^{th} training iteration.

It is known from Fig. 3 that up to 40 training iterations is required for the convergence of the ACM scenario. Therefore, in the subsequent experiment, $\mathcal{N}_0 = 40$ is used in another Linear-based configuration denoted using Linear40. From Fig. 5, the HasKill rate of Linear40 continues to rise with more training iterations. However, unlike SRE20, Linear40 is still not able to achieve 100% HasKill rate. In fact, it is still lagging the SRE20 configuration for the larger portion of the learning process. This can only lead to a conclusion that linear depreciation of exploration rate may actually need more training iterations than what can be achieved using the proposed technique.

The trend of HasKill rate of Linear40 in the top plot of Fig. 5 can be explained by correlating it to the plot of exploration rate in the top plot of Fig. 6. For Linear40, the exploration rates is declining in tandem with the linearly decaying ϵ seen at the bottom plot of Fig. 6. There is a lack of correlation to the HasKill rate of Linear40 configuration. The linearly decaying ϵ reduces the probability of finding more optimal action policies at a constant rate. Therefore, regulating the exploration rate using the proposed SRE algorithm actually facilitates the discovery of the more optimal action policies.

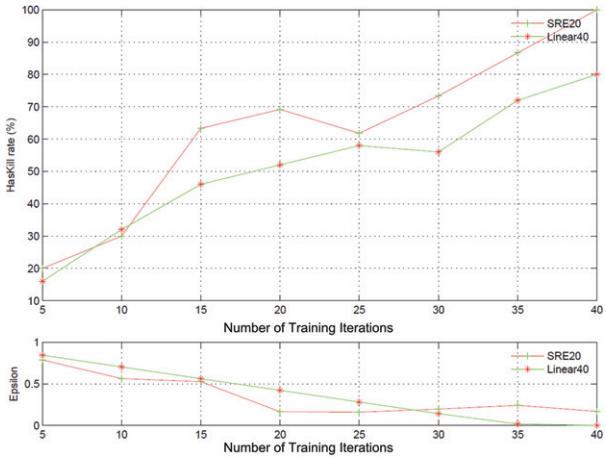

Fig. 5. Comparison of HasKill for the ACM scenario using $\mathcal{N}_0 = 40$ just for the Linear40 configuration

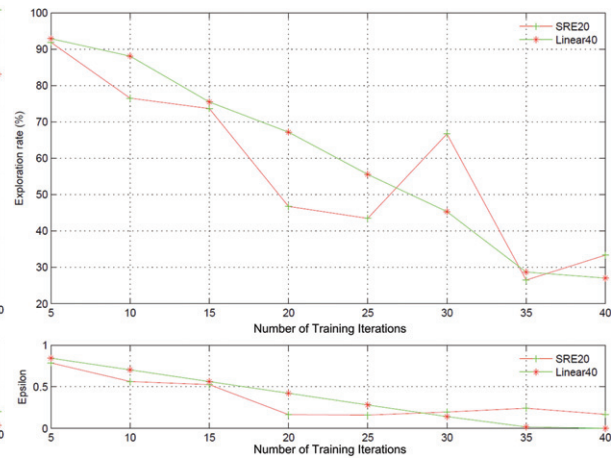

Fig. 6. Comparison of exploration rates for the ACM scenario using $\mathcal{N}_0 = 40$, $\epsilon_0 = 0.9$

5.2. The K-Armed Bandit problem domain

The K-Armed Bandit is a classical MDP problem domain used to investigate the balance between the exploration of the solution space and exploitation of the learned action policies [21]. It involves K slot machines that are randomly allocated with a fixed but unique payoff and it is only known to the players when it is pulled. The player is allowed to pull only $\mathcal{N}_p$ number of slot machines for each game where $\mathcal{N}_p < K$. The pull of a slot machine in each game cannot be repeated.

The payoff received by the player is accumulated after each pull of the slot machine in each game. The optimal payoff is known from the onset of the learning process. The goal of the player in each game is to achieve an accumulated payoff as close as possible to the optimal payoff. The player is allowed to improve on the accumulated payoff over multiple games using knowledge of the payoff of the slot machines learned from the previous games.

The interval success rate λ for the K-Armed Bandit is the accumulated payoff in terms of the percentage to the optimal payoff. For each game of the K-Armed Bandit, the player is expected to make 6 pulls on 15 slot machines, i.e. $\mathcal{N}_p = 6$ and $K = 15$. The plots for the K-Armed Bandit problem are aggregated from 20 runs of the experiments. Except for using $\rho_l^{c1} = 0.98$, $\mathcal{N}_w = 100$ and $\mathcal{N}_0 = 3000$, parameters for TD-FALCON and the SRE algorithm are as presented in Table 1 and Table 2.

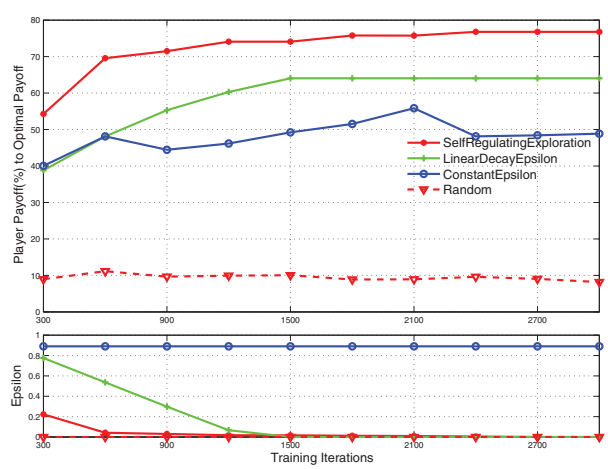

Fig. 7. Comparison of Player Payoff for the K-Armed Bandit problem using $\mathcal{N}_0 = 3000$

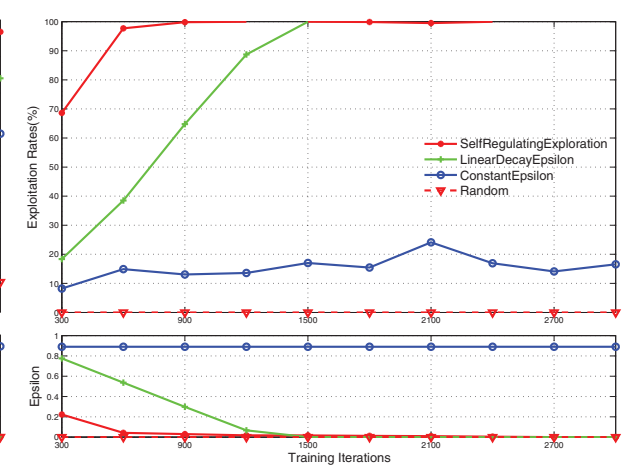

Fig. 8. Comparison of exploitation rates for the K-Armed Bandit problem using $\mathcal{N}_0 = 3000$

Comparisons of the Player Payoff using the proposed Self-Regulating Exploration techniques are made with the ϵ-greedy method with linear decay (LinearDecayEpsilon), constant ϵ method (ConstantEpsilon) and the Random approach. The top plot of Fig. 7 shows SelfRegulatingExploration outperforming all three other approaches from the onset of the learning process. The results of all three configurations are highly correlated to their ϵ shown at the bottom plot. In addition, it is also within expectation for LinearDecayEpsilon to outperform ConstantEpsilon.

This can be explained using plots of exploitation rates in Fig. 8. The SelfRegulatingEpsilon, LinearDecayEpsilon and ConstantEpsilon exploit the learned action policies at a probability of $(1 - \epsilon)$. Consequentially, LinearDecayEpsilon allows for full exploitation of the discovered action policies as ϵ is decayed fully. Keeping ϵ constant (at 0.90), exploitation rate of ConstantEpsilon fluctuates between 10% and 20%.

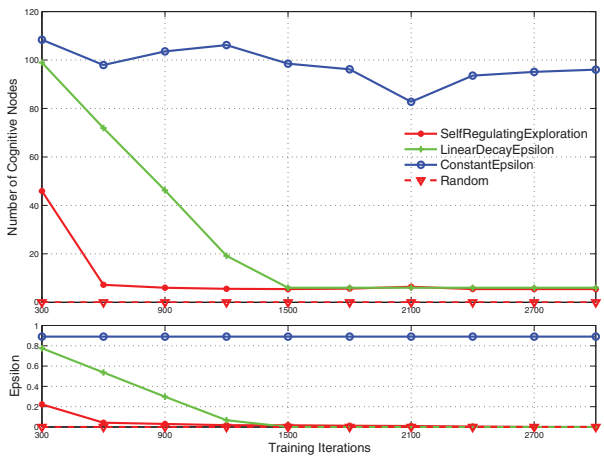

Fig. 9. Comparison of node population for the K-Armed Bandit problem using $\mathcal{N}_0 = 3000$

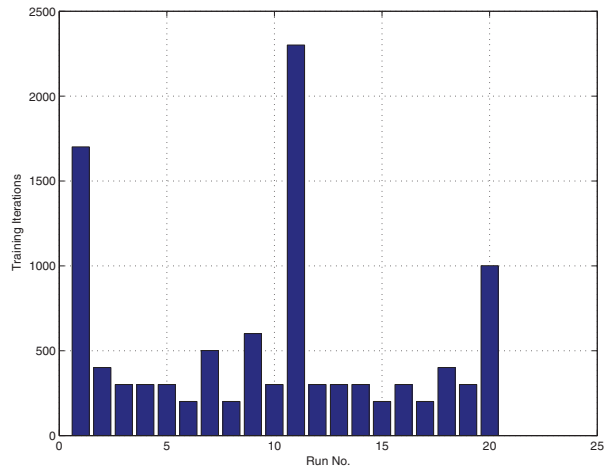

Fig. 10. Comparison of the actual number of training iterations for 20 runs for the SelfRegulatingExploration configuration

From the node population in Fig. 9, the SRE algorithm is also shown to improve learning efficiency. Using the pruning strategy presented in Section 3.3, the node population of SelfRegulatingExploration is pruned to the minimal level earlier than LinearDecayEpsilon. The node population of ConstantEpsilon fluctates about the 80 cognitive nodes level due to the constant exploration rate. Learning efficiency can be improved using the SRE algorithm because high λ that leads to high exploitation rates mean lesser cognitive nodes need to be learned. Early exploitation of the effective cognitive nodes leads to the pruning of the irrelevant cognitive nodes.

All configurations are initialized with 3000 training iterations. Unlike LinearDecayEpsilon, ConstantEpsilon and Random but similar to the results in Section 5.1, SelfRegulatingEpsilon uses (6) to fine-tune its remaining training duration $\mathcal{N}_r$. Therefore, the actual number of total training iteration for 20 runs of SelfRegulatingExploration are presented using Fig. 10. Together, this gives an average of 521 training iterations. This is a 82.63% reduction in the number of training iterations. This is not observed in any earlier works and will not be possible without using the SRE algorithm.

6. Conclusion

This work proposes a novel technique for self-regulating action exploration during reinforcement learning. It addresses the uncertainty over the amount of exploration required during learning and the total number of training iterations required for the learning process. Before, an informed estimation of the number of training iterations has to be made for each training session and it remained unchanged for the rest of the learning process. However, it is shown in this work that such estimations are often, at best, sub-optimal.

Using the proposed SRE algorithm, the exploration rate is regulated by correlating the interval success rate λ to the value of ϵ of the ϵ-greedy method. In addition, the updated ϵ is also used to derive the *remaining* number of training iterations $\mathcal{N}_r$. Adapting the length of the learning process to the interval success rate λ ensures just the

right number of training iterations for each run of the experiment. It is important to state that the proposed SRE algorithm assumes the presence of optimal action policies.

Experiments are conducted using a 1-v-1 ACM scenario and a standard reinforcement learning benchmark problem known as the K-Armed Bandit. Aggregated experimental results from these two multi-state MDP problem domains show tight correlation between the interval success rate λ and the exploration rate consistently lead to convergence using the right number of training iteration. For the ACM scenario, the right amount of training iterations is found to be more than the initial estimate of 20 training iterations. As for the K-Armed Bandit problem, the required number of training iteration is found to be around 82.63% lesser than the initial estimate of 3000 training iterations. From the experimental results, the SRE algorithm is shown to be a more effective and efficient approach than the standard approaches.

The SRE algorithm introduces two more degrees of autonomy to self-organizing neural networks such as FALCON. Now, it is able to exploit and explore in correlation to the status of the learning process and it is able to learn for as long as necessary. There are plans to conduct more in-depth investigations on how learning efficiency will change using $0.0 < \kappa < 1.0$ instead of $\kappa = 1.0$ and using windows of different sizes. Further demonstrations of the robustness of the SRE algorithm will also be conducted by presenting novel scenarios to FALCON at an ad-hoc basis during reinforcement learning. The ability of the SRE algorithm to self-regulate the exploration rate and the training duration should also facilitate the use of the FALCON in more challenging problem domains.

Acknowledgment

This work is supported by the DSO National Laboratories under research grant DSOCL11258. This project was conducted in close collaboration with Khee-Yin How and his team at DSO National Laboratories, Seng-Beng Ho and his team at Temasek Laboratories@NUS, Adrian Yeo and his team at CAE (S.E.A.) Pte. Ltd. and Sylvain Caron and his team at CAE (Montreal) Inc.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA: MIT Press, 1998.
[2] M. Tokic, Adaptive ϵ-greedy exploration in reinforcement learning based on value differences, LNCS-LNAI 6359 (2010) 203–210.
[3] K. Zhang, W. Pan, The two facets of the exploration-exploitation dilemma, in: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2007, pp. 371–377.
[4] S. Varges, G. Riccardi, S. Quarteroni, A. V. Ivanov, The exploration/exploitation trade-off in reinforcement learning for dialogue management, in: Proceedings of the IEEE Workshop on Automatic Speech Recognition Understanding, 2009, pp. 479–484.
[5] M. Kearns, S. Singh, Near-optimal reinforcement learning in polynomial time, Machine Learning 49 (2002) 209–232.
[6] L. Kaelbling, M. Littman, A. Moore, Reinforcement learning: A survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.
[7] A.-H. Tan, FALCON: A Fusion Architecture for Learning, Cognition, and Navigation, in: Proceedings of the IJCNN, 2004, pp. 3297–3302.
[8] T.-H. Teng, A.-H. Tan, Cognitive agents integrating rules and reinforcement learning for context-aware decision support, in: Proceedings of the IAT, 2008, pp. 318–321.
[9] D. Wang, B. Subagdja, A.-H. Tan, G.-W. Ng, Creating human-like autonomous players in real-time first person shooter computer games, in: Proceedings of the 21st Annual Conference on Innovative Applications of Artificial Intelligence, 2009, pp. 173–178.
[10] T.-H. Teng, A.-H. Tan, Y.-S. Tan, A. Yeo, Self-organizing Neural Networks for Learning Air Combat Maneuvers, in: Proceedings of the IJCNN, 2012, pp. 2859–2866.
[11] J. C. Gittins, Bandit processes and dynamic allocation indices, Journal of the Royal Statistical Society. Series B (Methodological) 41 (2) (1979) 148–177.
[12] N. Meuleau, P. Bourgine, Exploration of multi-state environments: local measures and back-propagation of uncertainty, Machine Learning 35 (2) (1999) 117–154.
[13] R. Patrascu, D. Stacey, Adaptive exploration in reinforcement learning, in: Proceedings of the IJCNN, Vol. 4, 1999, pp. 2276–2281.
[14] C. Domingo, Faster Near-Optimal Reinforcement Learning: Adding Adaptiveness to the E^3 Algorithm, in: O. Watanabe, T. Yokomori (Eds.), Algorithmic Learning Theory, Vol. 1720 of LNCS, Springer Berlin / Heidelberg, 1999, pp. 241–251.
[15] G. A. Carpenter, S. Grossberg, D. B. Rosen, Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system, Neural Networks 4 (1991) 759–771.
[16] A.-H. Tan, N. Lu, X. Dan, Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning with Delayed Evaluative Feedback, IEEE Transactions on Neural Networks 19 (2) (2008) 230–244.
[17] A.-H. Tan, Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning, in: Proceedings of the IJCAI, 2007, pp. 1071–1076.
[18] C. J. C. H. Watkins, P. Dayan, Q-Learning, Machine Learning 8 (3) (1992) 279–292.
[19] T.-H. Teng, Z.-M. Tan, A.-H. Tan, Self-organizing neural models integrating rules and reinforcement learning, in: Proceedings of the IJCNN, 2008, pp. 3770–3777.

[20] M. D. Ardema, N. Rajan, An approach to three-dimensional aircraft pursuit-evasion, Computers & Mathematics with Applications 13 (1-3) (1987) 97–110.
[21] H. Robbins, Some aspects of the sequential design of experiments, Bulletin of the American Mathematical Society 58 (1952) 527–535.