A Comparison of Continuous and Discretized Pursuit Learning Schemes

B. John Oommen* and Mariana Agache
School of Computer Science
Carleton University
Ottawa, K1S 5B6, Canada
oommen@scs.carleton.ca

ABSTRACT

A Learning Automaton is an automaton that interacts with a random environment, having as its goal the task of learning the optimal action based on its acquired experience. Many learning automata have been proposed, with the class of Estimator Algorithms being among the fastest ones. Thathachar and Sastry [23], through the Pursuit Algorithm, introduced the concept of learning algorithms that pursue the current optimal action, following a Reward-Penalty learning philosophy. Later, Oommen and Lanctôt [16] extended the Pursuit Algorithm into the discretized world by presenting the Discretized Pursuit Algorithm, based on a Reward-Inaction learning philosophy. In this paper, we argue that the Reward-Penalty and Reward-Inaction learning paradigms in conjunction with the continuous and discrete models of computation lead to four versions of Pursuit Learning Automata. We contend that a scheme that merges the Pursuit concept with the most recent response of the Environment permits the algorithm to utilize the LA's long-term and short-term perspectives of the Environment. In this paper, we present all the four resultant Pursuit algorithms, and also present a quantitative comparison between them. Although the present comparison is solely based on rigorous experimental results, we are currently investigating a formal convergence analysis of the various schemes.

1. INTRODUCTION

The goal of many intelligent problem-solving systems is to be able to make decisions without a complete knowledge of the consequences of the various choices available. In order for a system to perform well under conditions of uncertainty, it has to be able to acquire some knowledge about the consequences of different choices. This acquisition of the relevant knowledge can be expressed as a learning problem. In the quest to solve the learning problem, Tsetlin, a Russian mathematician, created in 1961 a new model of computer learning, which is now called a Learning Automaton (LA). The goal of such an automaton is to determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded. The functionality of the learning automaton can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with the environment. During a cycle, the automaton chooses an action, which triggers a response from the environment, a response that can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. By learning to choose the optimal action, the automaton adapts itself to the environment.

Learning automata have found applications in systems that posses incomplete knowledge about the environment in which they operate, such as game playing [1], [2], [3], pattern recognition [10], [20], object partitioning [17], [18]. They also have been applied to systems that have time varying environments, such as telephony routing [11], [12], and priority assignments in a queuing system [7]. The varieties of learning automata and their applications have been reviewed by Lakshmivarahan [2], and by Narendra and Thathachar [9]. Consequently, in this paper only a brief classification will be presented.

In the first LA designs, the transition and the output functions were time invariant, and for this reason these LA were considered "fixed structure" automata. Tsetlin, Krylov, and Krinsky [24], [25] presented notable examples of this type of automata. Later, Varshavskii and Vorontsova in [26] introduced a class of stochastic automata known in literature as Variable Structure Stochastic Automata (VSSA). These automata are characterized by the fact that the state transition probabilities or the action selecting probabilities are updated with time. For such an automaton, there is a bi-directional correspondence between the set of states and the set of actions of the automaton, each state corresponding to a particular action. Consequently, the set of states becomes redundant in the definition of a VSSA, and hence, the learning automaton is completely defined by a set of actions (which is the output of the automata), a set of inputs (which is usually the response of the environment) and a learning algorithm T. The learning algorithm operates on a probability vector

$$\mathbf{P}(t) = [p_1(t),\ldots,p_r(t)]^T,$$

where $p_i(t)$ (i = 1,…,r) is the probability that the automaton will select the action α_i at the time t:

$$p_i(t)=\Pr[\alpha(t)= \alpha_i],\ i=1,\ldots,r, \text{ and it satisfies,}$$

$$\sum_{i=1}^{r} p_i(t) = 1 \text{ for all 't'.}$$

This vector is known in the literature as the *Action Probability vector*. Moreover, VSSA are completely defined by a set of action probability updating rules operating on the action probability vector **P**(t) [2], [4], [9].

***Definition* 1**: A *Variable Structure Stochastic Automaton* (VSSA) is a 4-tuple <A,B,T,P>, where A is the set of actions, B is the set of inputs of the automaton (the set of outputs of the environment), and $T:[0,1]^r \times A \times B \rightarrow [0,1]^r$ is an updating scheme such that

$$\mathbf{P}(t+1) = T(\mathbf{P}(t), \alpha(t), \beta(t)), \qquad (1)$$

where P(t) is the Action Probability vector defined above, $\alpha(t)$ is the action chosen at time 't', and $\beta(t)$, is the response it has obtained.

In general, FSSA and the VSSA can be analyzed using the theory of Markov chains. For the VSSA, if the mapping T is independent of time, the probability **P**(t+1) is determined completely by **P**(t), which implies that $\{\mathbf{P}(t)\}_{t\geq 0}$ is a discrete-homogenous Markov process. From this perspective, different mappings, T, can identify different types of learning algorithms. If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as *absorbing algorithm*. Families of VSSA that posses absorbing barriers have been studied in [4], [8], [10]. Ergodic VSSA have also been investigated [2], [10], [13]. These VSSA converge in distribution and thus the asymptotic distribution of the action

* Senior Member, IEEE.

probability vector has a value that is independent of the corresponding initial vector. Because of this independence property, the ergodic VSSA are suitable for non-stationary environments. In stationary environments, automata with absorbing barriers may be preferred because they can be made to converge to the optimal action with a probability as close to unity as desired.

In practice, the relatively slow rate of convergence of these algorithms constituted a limiting factor in their applicability. In order to increase their speed of convergence, the concept of discretizing the probability space was introduced in [21]. This concept is implemented by restricting the probability of choosing an action to a finite number of values in the interval [0,1]. If the values allowed are equally spaced in this interval, the discretization is said to be linear, otherwise, the discretization is called non-linear. Following the discretization concept, many of the continuous VSSA have been discretized; indeed, various discrete automata have been presented in literature [13], [15].

In the quest to design faster converging learning algorithms, Thathachar and Sastry [22] opened another avenue by introducing a new class of algorithms, called "Estimator" Algorithms. The main feature of these algorithms is that they maintain running estimates for the reward probability of each possible action, and use them in the probability updating equations. Typically, in the first step of the functional cycle the automaton chooses an action and the environment generates a response to this action. Based on this response, the estimator algorithm updates the estimate of the reward probability for that action. The change in the action probability vector is based on both the running estimates of the reward probabilities, and on the feedback received from the environment. A detailed description of the estimator algorithms can be found in [5], [6], [16], [22], [23].

Contribution of this paper

Pursuit algorithms are a subset of the estimator algorithms. As their names suggest, these algorithms are characterized by the fact that the action probability vector "pursues" the action that is currently estimated to be the optimal action. This is achieved by increasing the probability of the action whose current estimate of being rewarded is maximal [16], [23].

The estimator algorithms presented until now update the probability vector $\mathbf{P}(t)$ based on the *long-term* properties of the Environment, and no consideration is given to a *short-term* perspective. In contrast, the VSSA and the FSSA rely only on the *short-term* (most recent responses) properties of the Environment for updating the probability vector $\mathbf{P}(t)$.

Besides these methods of incorporating the acquired knowledge into the probability vector, another important characteristic of the learning automata is represented by the philosophy of the learning paradigm. For example, by giving more importance to rewards than to penalties in the probability updating rules, the learning automata can considerably improve their convergence properties. In the case of the linear schemes of the VSSA, by updating the probability vector $\mathbf{P}(t)$ only if the environment rewarded the chosen action, the linear scheme L_{RI} became ε-optimal, whereas the symmetric linear Reward-Penalty scheme, L_{RP}, is at most expedient. Also, by considerably increasing the value of probability changes on reward in comparison to changes made on penalty, yields a resultant linear scheme, the $L_{R-\varepsilon P}$, which is ε-optimal. The same behavior can be observed in the case of the FSSA. The difference between the Krinsky automaton and the Tsetlin automaton is that the Krinsky automaton gives more importance to the rewards than to the penalties. This modification improves the performance of the Krinsky automaton, making it ε-optimal in all stationary environments, whereas the Tsetlin automaton is ε-optimal only in the environments where $\min\{c_1,c_2\} < 0.5$ [24].

In this paper, we argue that the automaton can model the *long term* behaviour of the Environment by maintaining running estimates of the reward probabilities. Additionally, we contend that the *short term* perspective of the Environment is also valuable, and we maintain that this information resides in the most recent responses that are obtained by the automaton. The paper presents schemes by which both the *short-term* and *long-term* perspectives of the Environment can be incorporated in the learning process – the long term information crystallized in terms of the running reward-probability estimates, and the short term information used by considering the whether the most recent response was a reward or a penalty. Thus, when *short-term* perspectives are considered, the Reward-Inaction and the Reward-Penalty learning paradigms become pertinent *in the context of the estimator algorithms*.

The Pursuit algorithm presented by Thathachar and Sastry in [23] considered only the long-term estimates in the probability updating rules. Later, Oommen and Lanctôt [16] presented a discretized version of a Pursuit algorithm which considered both the short-term and the long-term perspectives and embraced the *Reward-Inaction* learning paradigm. In this paper, we shall present these two Pursuit algorithms and new versions of Pursuit algorithms, which basically emerge from the combination of these learning "philosophies" and paradigms. Also, based on experimental results, a comparison of the rate of convergence of these algorithms will be presented.

2. PURSUIT ALGORITHMS

In this section we shall describe the two existing Pursuit algorithms, a continuous version introduced by Thathachar and Sastry and a discretized version, presented by Oommen and Lanctôt.

The Continuous Pursuit Reward-Penalty (CP_{RP}) Algorithm

The pioneering Pursuit algorithm, the Continuous Pursuit algorithm, was introduced by Thathachar and Sastry [23]. We present it here in all brevity. As alluded to earlier, the algorithm is based on the long-term estimates. In other words, it did not take into account the short term information (the most recent response), and thus modified the action probability vector at every time instant, yielding a Pursuit algorithm operating on a *Reward-Penalty* learning paradigm. For this reason, we shall refer to it as the Continuous Pursuit Reward-Penalty (CP_{RP}) algorithm. The CP_{RP} algorithm involves three steps [23]. The first step consists of choosing an action $\alpha(t)$ based on the probability distribution $\mathbf{P}(t)$. Whether the automaton is rewarded of penalized, the second step is to increase the component of $\mathbf{P}(t)$ whose reward estimate is maximal (the current optimal action), and to decrease the probability of all the other actions. Vectorially, the probability updating rules can be expressed as follows:

$$\mathbf{P}(t+1) = (1-\lambda)\,\mathbf{P}(t) + \lambda\,\mathbf{e}_m \qquad (2)$$

where $\mathbf{e}_m$ is the action which is currently estimated to be the "best" action.

This equation shows that the action probability vector $\mathbf{P}(t)$ is moved in the direction of the action with the current maximal reward estimate.

The last step is to update the running estimates for the probability of being rewarded. For calculating the vector with the reward estimates denoted by $\hat{\mathbf{d}}(t)$, two more vectors are introduced: $\mathbf{W}(t)$ and $\mathbf{Z}(t)$, where $Z_i(t)$ is the number of times the i^{th} action has been chosen and $W_i(t)$ is the number of times the action α_i has been rewarded. Formally, the algorithm can be described as follows.

ALGORITHM CP_{RP}

Parameters

λ the speed of learning parameter , where $0<\lambda<1$.

m index of the maximal component of the reward estimate vector

$\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,..,r} \{\hat{d}_i(t)\}$.

e_m unit r-vector with 1 in the m^{th} coordinate

$W_i(t)$ the number of times the i^{th} action has been rewarded up to time t, for $1\le i\le r$.

$Z_i(t)$ the number of times the i^{th} action has been chosen up to time t, for $1 \le i\le r$.

Method

Initialize $p_i(t)=1/r$, for $1\le i\le r$

Initialize $\hat{\mathbf{d}}(t)$ by choosing each action a small number of times.

Repeat

Step 1: At time t pick $\alpha(t)$ according to probability distribution P(t). Let $\alpha(t)= \alpha_i$.

Step 2: If α_m is the action with the current highest reward estimate, update P(t) as :

$$P(t+1) = (1-\lambda) P(t) + \lambda e_m \quad (3)$$

Step 3: Update $\hat{\mathbf{d}}(t)$ according to the following equations for the action chosen:

$$\begin{aligned} W_i(t+1) &= W_i(t) + (1-\beta(t)) \\ Z_i(t+1) &= Z_i(t) + 1 \\ \hat{d}_i(t+1) &= \frac{W_i(t+1)}{Z_i(t+1)} \end{aligned} \quad (4)$$

End Repeat

END ALGORITHM CP_{RP}

The CP_{RP} algorithm is similar in design to the L_{RP} algorithm, in the sense that both algorithms modify the action probability vector **P**(t) if the response from the environment is a reward or a penalty. The difference occurs in the way they approach the solution; whereas the L_{RP} algorithm moves **P**(t) in the direction of the most recently rewarded action or in the direction of all the actions not penalized, the CP_{RP} algorithm moves **P**(t) in the direction of the action which has the highest reward estimate.

Thathachar and Sastry in [23] proved that this algorithm is ε-optimal in any stationary random environment. In the context of this paper, we shall merely outline the proof of the convergence of this algorithm. Indeed, they proved the convergence in two stages. First, they showed that using a sufficiently small value for the learning parameter λ, all actions are chosen enough number of times so that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. This is stated in Theorem 1 below.

Theorem 1: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the CP_{RP} algorithm, for all $\lambda \in (0, \lambda^*)$,

Pr[All actions are chosen at least M times each before time t] > 1-δ, for all $t \geq t_0$.

The detailed proof for this result can be found in [23] ♦♦♦

The second stage of the proof of convergence of the CP_{RP} algorithm consists of showing that if there is such an action α_m, for which the reward estimate remains maximal after a finite number of iterations, then the m^{th} component of the action probability vector converges in probability to 1.

Theorem 2: Suppose that there exists an index *m* and a time instant $t_0 < \infty$ such that

$$\hat{d}_m(t) > \hat{d}_j(t), \ (\forall j)\ j \neq m, (\forall t)\ t > t_0 .$$

Then $p_m(t) \to 1$ with probability 1 as $t \to \infty$.

Sketch of Proof: To prove this result, we define the following the interval:

$$\Delta p_m(t) = E[p_m(t+1) - p_m(t) \mid Q(t)].$$

Using the assumptions of the theorem, this quantity becomes:

$$\Delta p_m(t) = \lambda(1 - p_m(t)) \geq 0, \text{for all } t \geq t_0 ,$$

which implies that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [9], $\{p_m(t)\}_{t \geq t_0}$ converges as $t \to \infty$,

$$E[p_m(t+1) - p_m(t) \mid Q(t)] \xrightarrow[t \to \infty]{} 0 \quad \text{with probability 1.}$$

Hence, $p_m(t) \to 1$ with probability 1. ♦♦♦

The final theorem that shows the ε-optimal convergence of the CP_{RP} algorithm can be stated as:

Theorem 3: For the CP_{RP} algorithm, in every stationary random environment, there exists $\lambda^* > 0$ and $t_0 > 0$, such that for all $\lambda \in (0, \lambda^*)$ and for any $\delta \in (0, 1)$ and any $\varepsilon \in (0, 1)$,

$$Pr[p_m(t) > 1 - \varepsilon] > 1 - \delta$$

for all $t > t_0$.

Sketch of Proof: The proof for this theorem can be easily deduced by the first two results. ♦♦♦

The Discretized Pursuit Reward-Inaction (DP_{RI}) Algorithm

In 1990, Oommen and Lanctôt introduced [16] a discretized version of a Pursuit algorithm. Apart from the long-term perspective of the Environment (recorded in the estimates of the reward probabilities), it also utilized the short-term perspective of the Environment that was modelled in terms of the most recent Environment response. This Pursuit algorithm was based on the *Reward-Inaction* learning paradigm. In the context of this paper, we shall refer to this algorithm as the Discretized Pursuit Reward-Inaction (DP_{RI}) Scheme. The differences between the discrete and continuous versions of the Pursuit algorithms occur only in the updating rules for the action probabilities, the second step of the algorithm. The discrete Pursuit algorithms make changes to the probability vector **P**(t) in discrete steps, whereas the continuous versions use a continuous function to update **P**(t).

In the DP_{RI} algorithm, when an action is rewarded, all the actions that do not correspond to the highest estimate are decreased by a step Δ, where Δ=1/rN, and N is a resolution parameter. In order to keep the sum of the components of the vector **P**(t) equal to unity, the probability of the action with the highest estimate has to be increased by an integral multiple of the smallest step size Δ. When the action chosen is penalized, there is no update in the action probabilities, and it is thus of the *Reward-Inaction* paradigm. This, in principle, fully describes the algorithm, given formally below.

ALGORITHM DP_{RI}

Parameters

m index of the maximal component of the reward estimate vector

$\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,\ldots,r} \{\hat{d}_i(t)\}$.

$W_i(t)$ the number of times the i^{th} action has been rewarded up to time t, for $1 \leq i \leq r$.

$Z_i(t)$ the number of times the i^{th} action has been chosen up to time t, for $1 \leq i \leq r$.

N resolution parameter

$\Delta = 1/rN$ is the smallest step size

Method

Initialize $p_i(t) = 1/r$, for $1 \leq i \leq r$

Initialize $\hat{\mathbf{d}}(t)$ by choosing each action a small number of times.

Repeat

Step 1: At time t pick $\alpha(t)$ according to probability distribution **P**(t). Let $\alpha(t) = \alpha_i$.

Step 2: Update **P**(t) according to the following equations:

If $\beta(t)=0$ and $p_m(t) \neq 1$ Then

$$\begin{aligned} p_j(t+1) &= \max\{p_j(t) - \Delta, 0\} \\ p_m(t+1) &= 1 - \sum_{j \neq m} p_j(t+1) \end{aligned} \quad (5)$$

Else

$$p_j(t+1) = p_j(t) \text{ for all } 1 \le j \le r. \quad (6)$$

Step 3: Update $\hat{\mathbf{d}}(t)$ exactly as in the CP_{RP} Algorithm

End Repeat

END ALGORITHM DP_{RI}

Oommen and Lanctôt proved that this algorithm satisfies both the properties of moderation and monotonically [16] required for any discretized "Estimator" algorithm to converge. They also showed that the algorithm is ε-optimal in every stationary random environment.

The proof of the convergence of this algorithm follows the same trend as the proof for the Pursuit CP_{RI} algorithm, with the necessary adjustments made to accommodate for the discretization of the probability space [0,1]. Thus, Oommen and Lanctôt proved that if the m^{th} action is rewarded more than any action from time t_0 onward then the action probability vector for the DP_{RI} will converge to the unit vector $\mathbf{e}_m$. These results are stated below.

Theorem 4: Suppose there exists an index m and a time instant $t_0<\infty$ such that $\hat{d}_m(t) > \hat{d}_j(t)$ for all j such that $j \neq m$ and all $t \ge t_0$. Then there exists an integer N_0 such that for all resolution parameters $N>N_0$, $p_m(t) \to 1$ with probability 1 as $t \to \infty$.

Sketch of Proof: The proof for this theorem aims to show that $\{p_m(t)\}_{t \ge t_0}$ is a submartingale satisfying $\sup_{t \ge 0} E[\| p_m(t) \|] < \infty$. Then, based on the submartingale convergence theorem [9] $\{p_m(t)\}_{t \ge t_0}$ converges, and so,

$$E[p_m(t+1) - p_m(t) \mid Q(t)] \xrightarrow[t \to \infty]{} 0 .$$

Indeed, the authors of [16] showed that

$$E[p_m(t+1) \mid Q(t), p_m(t) \neq 1] = p_m(t) + d_m c_t \Delta ,$$

where c_t is an integral, bounded by 0 and r, such that $p_m(t+1) = p_m(t) + c_t \Delta$. Thus,

$$E[p_m(t+1) - p_m(t) \mid Q(t)] = d_m c_t \Delta \ge 0 \text{ , for all } t \ge t_0,$$

implying that $p_m(t)$ is a submartingale. From the submartingale convergence theorem they infer that $d_m c_t \Delta \to 0$ with probability 1. This in turn implies that $c_t \to 0$ w.p. 1, and consequently, that $\sum_{j \neq m} \max(p_j(t) - \Delta, 0) \to 0$ w.p. 1. Hence $p_m(t) \to 1$ w.p. 1. ♦♦♦

The next step in proving the convergence of this algorithm is to show that using a sufficiently large value for the resolution parameter N, all actions are chosen enough number of times so that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time.

Theorem 5: For each action α_i, assume $p_i(0) \neq 0$. Then for any given constants $\delta > 0$ and $M < \infty$, there exists $N_0 < \infty$ and $t_0 < \infty$ such that under DP_{RI}, for all learning parameters $N > N_0$ and all time $t > t_0$:

Pr{each action chosen more than M times at time t} $\ge 1-\delta$.

The proof of this theorem is similar to the proof of the Theorem 1, and can be found in [16]. ♦♦♦

These two theorems lead to the result that the DP_{RI} scheme is ε-optimal in all stationary random environments.

3. NEW ALGORITHMS

Applying different learning paradigms to the principle of pursuing the action with the best reward estimate leads to four learning algorithms. These are listed below:

Algorithm DP_{RI}: Discretized Pursuit Reward-Inaction Scheme

Paradigm: Reward-Inaction; **Probability Space**: Discretized

Algorithm DP_{RP}: Discretized Pursuit Reward-Penalty Scheme

Paradigm: Reward- Penalty; **Probability Space**: Discretized

Algorithm CP_{RI}: Continuous Pursuit Reward-Inaction Scheme

Paradigm: Reward-Inaction; **Probability Space**: Continuous

Algorithm CP_{RP}: Continuous Pursuit Reward- Penalty Scheme

Paradigm: Reward- Penalty; **Probability Space**: Continuous

Observe that of the above four, algorithms CP_{RP} and DP_{RI} were already presented in the literature and were described in the previous section. The algorithms that are now new to the field of Pursuit schemes are the CP_{RI} and DP_{RP} mentioned above. We shall discuss these briefly below.

The Continuous Pursuit Reward-Inaction (CP_{RI}) Algorithm

The continuous Pursuit algorithm based on the reward-inaction learning "philosophy" represents a continuous version of the discretized algorithm of Oommen and Lanctôt in [16]. The algorithm differs from the CP_{RP} algorithm only in its updating probability rules. As before, the long-term perspective of the Environment is recorded in the estimates of the reward probabilities. But it utilizes the short-term perspective of the Environment by updating the probability vector based on the most recent Environment response. Being a Reward-Inaction algorithm, it updates the action probability vector $\mathbf{P}(t)$ only if the current action is rewarded by the environment. If the action is penalized, the action probability vector remains unchanged. The algorithm follows.

ALGORITHM CP_{RI}

Parameters

λ, m, $\mathbf{e}_m$, $W_i(t)$, $Z_i(t)$: Same as in the CP_{RP} algorithm.

Method

Initialize $p_i(t)=1/r$, for $1 \le i \le r$

Initialize $\hat{\mathbf{d}}(t)$ by choosing each action a small number of times.

Repeat

Step 1: At time t pick $\alpha(t)$ according to probability distribution $\mathbf{P}(t)$. Let $\alpha(t)= \alpha_i$.

Step 2: If α_m is the action with the current highest reward estimate, update $\mathbf{P}(t)$ as :

If $\beta(t)=0$ Then

$$\mathbf{P}(t+1) = (1-\lambda)\,\mathbf{P}(t) + \lambda\,\mathbf{e}_m \quad (7)$$

Else $\mathbf{P}(t+1)=\mathbf{P}(t)$

Step 3: Update $\hat{\mathbf{d}}(t)$ exactly as in the CP_{RP} Algorithm

End Repeat

END ALGORITHM CP_{RI}

Like the CP_{RP}, the CP_{RI} algorithm can be proved ε-optimal in any stationary environment. The proof for the ε-optimality of the CP_{RI} follows the same idea as the other Pursuit algorithms. First, it can be shown that using a sufficiently small value for the learning parameter λ, all actions are chosen enough number of times so that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. Mathematically, this can be expressed with the following result.

Theorem 6: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the CP_{RI} algorithm, for all $\lambda \in (0, \lambda^*)$,

Pr[All actions are chosen at least M times each before time t] $> 1-\delta$, for all $t \ge t_0$. ♦♦♦

The proof for this theorem is very similar to the proof for the Theorem 1, and so we omit the details here. Furthermore, it can be shown (Theorem 7) that if there is an action α_m, for which the reward estimate remains maximal after a finite number of iterations, then the m^{th} component of the action probability vector converges in probability to 1. Finally, Theorem 8 expresses the ε-optimality convergence for the CP_{RI} algorithm, and can be easily deducted from the two previous results. The proof of these theorems is remarkable similar to that of Theorem 2 and 3, and is omitted.

Theorem 7: Suppose that there exists an index *m* and a time instant $t_0 < \infty$ such that

$$\hat{d}_m(t) > \hat{d}_j(t), \ (\forall j)\ j \neq m, (\forall t)\ t > t_0 .$$

Then $p_m(t) \rightarrow 1$ with probability 1 as $t \rightarrow \infty$. ♦♦♦

Theorem 8: For the CP_{RI} algorithm, in every stationary random environment, there exists a $\lambda^* > 0$ and $t_0 > 0$, such that for all $\lambda \in (0, \lambda^*)$ and for any $\delta \in (0, 1)$ and any $\varepsilon \in (0, 1)$,

$$\Pr[p_m(t) > 1 - \varepsilon] > 1 - \delta \quad \text{for all } t > t_0. \qquad ♦♦♦$$

The Discretized Pursuit Reward-Penalty (DP_{RP}) Algorithm

By combining the strategy of discretizing the probability space and the Reward-Penalty learning paradigm in the context of a Pursuit "philosophy", we shall present another version of discrete Pursuit algorithm, denoted by DP_{RP} above. As any discrete algorithm, the DP_{RP} algorithm makes changes to the action probability vector in discrete steps. If N is a resolution parameter, the quantity that the components of **P**(t) can change by is given by multiples of Δ, where Δ=1/rN, r is the number of actions and N is a resolution parameter. Formally, the algorithm can be expressed as follows.

ALGORITHM DP_{RP}

Parameters

m, $W_i(t)$, $Z_i(t)$, N and Δ : Same as in the DP_{RI} algorithm.

Method

Initialize $p_i(t)=1/r$, for $1 \le i \le r$

Initialize $\hat{\mathbf{d}}(t)$ by choosing each action a small number of times.

Repeat

Step 1: At time t pick $\alpha(t)$ according to probability distribution **P**(t). Let $\alpha(t) = \alpha_i$.

Step 2**:** Update **P**(t) according to the following equations:

If $p_m(t) \neq 1$ Then

$$p_j(t+1) = \max_{j \neq m}\{p_j(t) - \Delta, 0\}$$
$$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1) \qquad (8)$$

Step 3: Update $\hat{\mathbf{d}}(t)$ exactly as in the DP_{RI} Algorithm

End Repeat

END ALGORITHM DP_{RP}

Just as in the case of the DP_{RI} algorithm, we can show that DP_{RP} algorithm is ε-optimal in every stationary environment. The proof is very similar to the proof for convergence of the DP_{RI} and is omitted to avoid repetition and in the interest of brevity.

4. EXPERIMENTAL RESULTS

Having introduced the various possible forms of continuous and discretized Pursuit algorithms, we shall now compare them experimentally. Indeed, in order to compare their relative performances, we performed simulations to accurately characterize their respective rates of convergence. In all the tests performed, an algorithm was considered to have converged if the probability of choosing an action was greater or equal to a threshold T ($0 < T \le 1$). If the automaton converged to the best action (*i.e.*, the one with the highest probability of being rewarded), it was considered to have converged correctly.

Before comparing the performance of the automata, innumerable multiple tests were executed to determine the "best" value of the respective learning parameters for each individual algorithm. The value was reckoned as the "best" value if it yielded the fastest convergence and the automaton converged to the correct action in a sequence of NE experiments. These best parameters were then chosen as the final parameter values used for the respective algorithms to compare their rates of convergence[1].

When the simulations were performed considering the same threshold T and number of experiments, NE, as Oommen and Lanctôt did in [16], (*i.e.* T=0.99 and NE=75), the learning parameters obtained for the DP_{RI} algorithm in the (0.8 0.6) environment had a variance coefficient of 0.2756665 in 40 tests performed. This variance coefficient was not considered satisfactory for comparing the performance of the Pursuit algorithms. Subsequent simulations were performed imposing stricter convergence requirements by increasing the threshold T, and proportionally, the number of experiments NE, which yielded learning parameters with smaller variance coefficients. For example, the learning parameter (N) for DP_{RI} algorithm in the (0.8 0.6) environment, when T=0.999 and NE=750, exhibits a variance coefficient of 0.0706893, which represents a much smaller variance. Therefore, in this paper the simulation results for T=0.999 and NE equal to 500 and 750 experiments shall be presented.

The simulations were performed for different existing benchmark environments with two and ten actions. These environments have been used also to compare a variety of continuous and discretized schemes, and in particular the DP_{RI} in [16] and to compare the performance of the CP_{RP} against other traditional VSSA in [23]. Furthermore, to keep the conditions identical, each estimator algorithm sampled all actions 10 times each in order to initialize the estimate vector. These extra iterations are also included in the results presented in the following tables. Table 1 and

Table 2 contain the simulation results for these four algorithms in two action environments. The probability of reward for one action was fixed at 0.8 for all simulations and the probability of reward for the second action was increased from 0.5 to 0.7[2]. In each case, the reported results correspond to the results obtained using the above-described "best" parameter.

---

[1] The reader will observe that there is a considerable difference between the results presented here and the results presented in [23]. In [23], the parameter chosen was the one which gave correct convergence in 25 parallel experiments. However, on testing the CP_{RP} for 1000 experiments, it was observed that it yielded only 84% accuracy. Thus, in the case of the CP_{RP} what we seek is the largest parameter, λ, which yields correct convergence *in all the 750 and 500 experiments respectively*. Similarly, in the case of the DP_{RP} we seek the smallest integer parameter, N, which yields correct convergence *in all the 750 and 500 experiments respectively*.

[2] When the reward probability for the second action is less than 0.5, the iterations required for convergence, after the initial 20, is very small (between 3 and 10), and do not permit meaningful comparison. They are thus omitted from the Table 1 and Table 2.

Table 1: Comparison of the Pursuit Algorithms in two-action benchmark environments for which exact convergence was required in 750 experiments (NE=750).[3]

Environment		DP_{RI}		DP_{RP}		CP_{RI}		CP_{RP}	
d_1	d_2	N	No. Iter.	N	No. Iter.	λ	No. Iter.	λ	No. Iter.
0.8	0.5	20	49	32	53	0.214	55	0.122	69
0.8	0.6	58	105	89	118	0.046	198	0.027	258
0.8	0.7	274	430	391	456	0.0091	939	0.0072	942

Table 2: Comparison of the Pursuit Algorithms in two-action benchmark environments for which exact convergence was required in 500 experiments (NE=500).

Environment		DP_{RI}		DP_{RP}		CP_{RI}		CP_{RP}	
d_1	d_2	N	No. Iter.	N	No. Iter.	λ	No. Iter.	λ	No. Iter.
0.8	0.5	17	44	26	47	0.314	43	0.169	55
0.8	0.6	52	97	74	102	0.054	171	0.036	199
0.8	0.7	217	357	297	364	0.011	789	0.0075	905

Table 3: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence 0was required in 750 experiments. The Reward probabilities for the actions are:

E_A: 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2
E_B: 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

Environment	DP_{RI}		DP_{RP}		CP_{RI}		CP_{RP}	
	N	No. Iter.	N	No. Iter.	λ	No. Iter.	λ	No. Iter.
E_A	188	752	572	1126	0.0097	1230	0.003	2427
E_B	1060	2693	1655	3230	0.002	4603	0.00126	5685

Table 4: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 500 experiments. The Action Reward probabilities are the same as in Table 3.

Environment	DP_{RI}		DP_{RP}		CP_{RI}		CP_{RP}	
	N	No. Iter.	N	No. Iter.	λ	No. Iter.	λ	No. Iter.
E_A	153	656	377	872	0.0128	970	0.0049	1544
E_B	730	2084	1230	2511	0.00225	4126	0.00128	5589

The simulations suggest that as the difference in the reward probabilities decreases (i.e., as the environment gets more difficult to learn from), the **Discretized** Pursuit algorithms exhibit a performance superior to the **Continuous** algorithms. Also, comparing the Pursuit algorithms based on the Reward-Inaction paradigm with the Pursuit algorithms based on the Reward-Penalty paradigm, one can notice that, in general, the Pursuit Reward-Inaction algorithms are up to 30% faster than the Reward-Penalty Pursuit algorithms. For example, when d_1=0.8 and d_2=0.6, the DP_{RI} converges to the correct action in an average of 105 iterations, and the DP_{RP} algorithm converges in an average of 118 iterations. In the same environment, the CP_{RI} algorithm takes an average of 198 iterations and the CP_{RP} requires 258 iterations, indicating that it is about 30% slower than the CP_{RI} algorithm.

To render the suite complete, similar experiments were also performed in the benchmark ten-action environments [16], [23]. The following table presents the results obtained in these environments.

As in the previous two-action environments, in ten-action environments, the DP_{RI} algorithm proved to have the best performance, converging to the correct action almost 25% faster than the DP_{RP} algorithm, and almost 50% faster than the CP_{RP} algorithm. If we analyze the behavior of these automata in the first environment, E_A, when NE=750, the average number of iterations required by the DP_{RI} to converge is 752, whereas the DP_{RP} requires 1126, which implies that the DP_{RI} algorithm is almost 30% faster than the DP_{RI}. In the same environment, the CP_{RI} requires an average of 1230 iterations for convergence and the CP_{RP} requires 2427, which shows that the CP_{RI} is 50% faster than CP_{RP}, and the DP_{RI} is almost 70% faster than the CP_{RP}.

Based on these experimental results, we can rank the various Pursuit algorithms in terms of their relative efficiencies - the number of iterations required to attain the same accuracy of convergence. The ranking is as follows:

Best Algorithm:
Discretized Pursuit Reward-Inaction (DP_{RI})
2nd-best Algorithm:
Discretized Pursuit Reward-Penalty (DP_{RP})
3rd-best Algorithm:
Continuous Pursuit Reward-Inaction (CP_{RI})
4th-best Algorithm:
Continuous Pursuit Reward-Penalty (CP_{RP})

Indeed, if we compare the algorithms quantitatively, we observe that the discretized versions are up to 30% faster than their continuous counterparts. Furthermore, if we compare the Reward-Inaction Pursuit algorithms against the Reward-Penalty algorithms, we see that the Reward-Inaction algorithms are superior in the rate of convergence, being up to 25% faster than their Reward-Penalty counterparts. Although it is clear that these comparative results are based only on the experimental results obtained from simulations, we believe that these results characterize the properties of the algorithms studied. A formal convergence analysis of the various schemes is currently being done.

5. CONCLUSION

Over the last decade, many new families of learning automata have emerged, with the class of Estimator Algorithms being among the fastest ones. Thathachar and Sastry [23], through the Pursuit Algorithm, introduced the concept of algorithms that pursue the current optimal action, following a *Reward-Penalty* learning philosophy. Later, Oommen and Lanctôt [16] extended the Pursuit Algorithm into the discretized world by presenting the Discretized Pursuit Algorithm, based on a *Reward-Inaction* learning philosophy. In this paper, we argued that a scheme that merges the Pursuit concept with the most recent response of the Environment permits the algorithm to utilize the LA's *long-term* and *short-term* perspectives of the Environment. Thus, we have demonstrated that the combination of the Reward-Penalty and Reward-Inaction learning paradigms in conjunction with the continuous and discrete models of computation, can lead to four versions of Pursuit Learning Automata. In this paper, we presented them all, (DP_{RI}, DP_{RP}, CP_{RI}, CP_{RP}) and also presented a quantitative comparison between them. Overall, the Discrete Pursuit Reward-Inaction algorithm surpasses the performance of all the other versions of Pursuit algorithms. Also, the Reward-Inaction schemes are superior to their Reward-Penalty counterparts.

6. ACKNOWLEDGMENTS

This work was partially supported by the National Science and Engineering Research Council of Canada.

[3] The number of iterations presented in these tables are rounded to the nearest integer.

The authors are grateful to Professor M. A. L. Thathachar, of the Indian Institute of Science, Bangalore, India, for his valuable comments and suggestions during the course of this study.

REFERENCES

[1] S. Baba, S. T. Soeda, and Y. Sawaragi, "An application of stochastic automata to the investment game", *Int. J. Syst. Sci*., Vol. 11, No. 12, pp. 1447-1457, Dec. 1980.

[2] S. Lakshmivarahan, *Learning Algorithms Theory and Applications*, New York: Springer-Verlag, 1981.

[3] S. Lakshmivarahan, "Two person decentralized team with incomplete information", *Appl. Math. and Computation*, Vol. 8, pp. 51-78, 1981.

[4] S. Lakshmivarahan and M. A. L. Thathachar, "Absolutely expedient algorithms for stochastic automata", *IEEE Trans. man. Cybern.*, Vol. SMC-3, pp. 281-286, 1973.

[5] J. K. Lanctôt, *Discrete Estimator Algorithms: A M athematical Model of Computer Learning*, M.Sc. Thesis, Dept. Math. Statistics, Carleton Univ., Ottawa, Canada, 1989.

[6] J. K. Lanctôt and B. J. Oommen, "Discretized Estimator Learning Automata", *IEEE Trans. on Syst. Man and Cybernetics*, Vol. 22, No. 6, pp. 1473-1483, November/December 1992.

[7] M. R. Meybodi, *Learning Automata and its Application to Priority Assignment in a Queuing System with Unknown Characteristic*, Ph.D. Thesis, School of Elec. Eng. and Computing Sci., Univ. Oklahoma, Norman, OK.

[8] K. S. Narendra and S. Lakshmivarahan, "Learning automata: A critique", *J. Cybern. Inform. Sci*., Vol. 1, pp. 53-66, 1987.

[9] K. S. Narendra and M. A. L. Thathachar, *Learning Automata*, Englewood cliffs, NJ, Prentice-Hall, 1989.

[10] K. S. Narendra and M. A. L. Thathachar, "Learning Automata – A Survey, *IEEE Trans. on Syst. Man and Cybernetics*, Vol. SMC-4, 1974, pp. 323-334.

[11] K. S. Narendra and M. A. L. Thathachar, "On the behavior of a learning automata in a changing environment with routing applications", *IEEE Trans. Syst. Man Cybern*., Vol. SMC-10. pp. 262-269, 1980.

[12] K. S. Narendra, E. Wright, and L. G. Mason, "Applications of Learning Automata to Telephone Traffic Routing", *IEEE Trans. Syst. Man. Cybern*., Vol. SMC-7, pp. 785-792, 1977.

[13] B. J. Oommen, "Absorbing and Ergodic Discretized Two-Action Learning Automata", *IEEE Trans. Syst. Man. Cybern.*, Vol. SMC-16, pp. 282-296, 1986.

[14] B. J. Oommen and J. R. P. Christensen, "Epsilon-Optimal Discretized Reward-Penalty Learning Automata", *IEEE. Trans. Syst. Man. Cybern.*, Vol. SMC-18, pp. 451-458, May/June 1988.

[15] B. J. Oommen and E.R. Hansen, "The Asymptotic Optimality of Discretized Linear Reward-Inaction Learning Automata", *IEEE. Trans. Syst. Man. Cybern*., pp. 542-545, May/June 1984.

[16] B.J. Oommen and J. K. Lanctôt, "Discretized Pursuit Learning Automata", *IEEE Trans. Syst. Man. Cybern*., vol. 20, No.4, pp.931-938, July/August 1990.

[17] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equi-partitioning problem", *IEEE Trans. Comput*., Vol. 37, pp. 2-14, Jan 1988.

[18] B. J. Oommen and D. C. Y. Ma, "Fast Object Partitioning Using Stochastic Learning Automata", in *Proc. 1987 Int. Conf. Research Development in Inform. Retrieval*, New Orleans, LA, June 1987.

[19] B. J. Oommen and M. A. L. Thathachar, "Multiaction Learning Automata Possessing Ergodicity of the Mean", *Inform. Sci*., vol. 35, no. 3, pp. 183-198, June 1985.

[20] R. Ramesh, *Learning Automata in Pattern Classification*", M.E. Thesis, Indian Institute of Science, Bungalore, India, 1983.

[21] M. A. L. Thathachar and B. J. Oommen, "Discretized Reward-Inaction Learning Automata", *J. Cybern. Information Sci*., pp. 24-29, Spring 1979.

[22] M. A. L. Thathachar and P.S. Sastry, "A Class of Rapidly Converging Algorithms for Learning Automata", presented at IEEE Int. Conf. on Cybernatics and Society, Bombay, India, Jan. 1984.

[23] M. A. L. Thathachar and P.S. Sastry, "Estimator Algorithms for Learning Automata", *Proc. Platinum Jubilee Conf. on Syst. Signal Processing*, Dept. Elec. Eng., Indian Institute of Science, Bangalore, India, Dec. 1986.

[24] M.L. Tsetlin, "On the Behavior of Finite Automata in Random Media", *Automat. Telemek.* (USSR), Vol. 22, pp. 1345-1354, Oct. 1961.

[25] M.L. Tsetlin, *Automaton Theory and the Modeling of Biological Systems*, New York: Academic, 1973.

[26] V.I. Varshavskii and I.P. Vorontsova, "On the Behavior of Stochastic Automata with Variable Structure", *Automat. Telemek.* (USSR), Vol. 24, pp. 327-333, 1963.