CONTINUOUS AND DISCRETIZED GENERALIZED PURSUIT LEARNING SCHEMES

Mariana Agache and B. John Oommen[1]

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 [24], through the Pursuit Algorithm, introduced the concept of learning algorithms. Their algorithm pursues only the current estimated optimal action. If this action is not the one with the minimum penalty probability, this algorithm pursues a wrong action. In this paper, we argue that a Pursuit scheme that generalizes the traditional Pursuit algorithm by pursuing all the actions with higher reward estimates than the chosen action, minimizes the probability of pursuing a wrong action, and is a faster converging scheme. To attest this, in this paper we present two new generalized Pursuit algorithms and also present a quantitative comparison of their performance against the existing Pursuit algorithms.

I. 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

[1]Senior Member IEEE. Address of the first two authors: School of Computer Science, Carleton University, Ottawa; Canada: K1S 5B6; e-mail address: oommen@scs.carleton.ca. The work of the second author was supported in part by the Natural Sciences and Engineering Research Council of Canada.

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], [21], and 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 definition of a Variable Structure Stochastic Automaton (VSSA), 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 $p_i(t)$ $(i = 1,\ldots,r)$ 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,…,r, 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*. VSSA are completely defined by a set of action probability updating rules operating on the action probability vector **P**(t) [2], [4], [9].

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 [22]. 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 [13], [15].

In the quest to design faster converging learning algorithms, Thathachar and Sastry [23] 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], [23], [24].

I.1. Contribution of this paper

Pursuit algorithms are a subset of the estimator algorithms. The existing Pursuit 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], [24]. This implies that if, at any time 't', the action that has the maximum reward estimate is not the action that has the minimum penalty probability, then the automaton pursues a wrong action. In an attempt to minimize this probability of pursuing a wrong action, our goal in this paper is to generalize the design of the Pursuit algorithm such that it pursues a set of actions. Specifically, these actions have higher reward estimates than the current chosen action. In this paper, we introduce two new Pursuit algorithms, a continuous and a discretized version, that use such a generalized learning approach.

II. PURSUIT ALGORITHMS

Thathachar and Sastry introduced the concept of Pursuit Algorithms [24] by presenting a continuous Pursuit algorithm that used a *Reward-Penalty* learning paradigm, denoted CP_{RP}. Later, in 1990, Oommen and Lanctôt [16] introduced the first discretized Pursuit estimator algorithm by presenting a discretized version, denoted DP_{RI}, that uses a Reward-Inaction learning paradigm. Oommen and Agache in [20] explored all Pursuit algorithms that resulted from the combination of the continuous and discrete probability space with the Reward-Penalty and Reward-Inaction learning paradigms, and introduced two new Pursuit algorithms, the Continuous Reward-Inaction Pursuit Algorithm (CP_{RI}) and the Discretized Reward-Penalty Pursuit Algorithm (DP_{RP}). In the interest of brevity, we present here only the CP_{RP} and the DP_{RI} algorithms.

II.1. The Continuous Pursuit Reward-Penalty (CP_{RP}) Algorithm

The pioneering Pursuit algorithm, the Continuous Pursuit algorithm, was introduced by Thathachar and Sastry [24]. We present it here in all brevity. The algorithm uses a *Reward-Penalty* learning paradigm, meaning that it updates the probability vector P(t) if the Environment rewards or penalizes the chosen action. For this reason, we shall refer to it as the Continuous Pursuit Reward-Penalty (CP_{RP}) algorithm. The CP_{RP} algorithm involves three steps [24]. 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:

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

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 $\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,..,r} \{\hat{d}_i(t)\}$.
$\mathbf{e}_m$	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 $\mathbf{P}(t)$. Let $\alpha(t)= \alpha_i$.

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

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

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

$$W_i(t+1) = W_i(t) + (1-\beta(t))$$
$$Z_i(t+1) = Z_i(t) + 1$$
$$\hat{d}_i(t+1) = \frac{W_i(t+1)}{Z_i(t+1)}$$

End Repeat

END ALGORITHM CP_{RP}

Thathachar and Sastry in [24] proved that this algorithm is ε-optimal in any stationary random environment. We shall merely state their essential analytic results.

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

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 $t > t_0$. ♦♦♦

II.2. The Discretized Pursuit Reward-Inaction (DP_{RI}) Algorithm

In 1990, Oommen and Lanctôt introduced [16] a discretized version of a Pursuit algorithm. This Pursuit algorithm was based on the *Reward-Inaction* learning paradigm, meaning that it updates the action probability vector **P**(t) only if the Environment rewards the chosen action, and even then, only in discrete steps. We refer to this algorithm as the Discretized Pursuit Reward-Inaction (DP_{RI}) Scheme.

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

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

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

N resolution parameter

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

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

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

$$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1)$$

Else

$p_j(t+1) = p_j(t)$ for all $1 \le j \le r$.

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.

III. GENERALIZED PURSUIT ALGORITHMS

The main idea that characterizes the existing Pursuit algorithms is that they *'pursue'* the best-estimated action, which is the action corresponding to the maximal estimate. In any iteration, these

algorithms increase only the probability of the best-estimated action, ensuring that the probability vector **P**(t) moves towards the solution that has the maximal estimate at the current time. This implies that if, at any time 't', the action that has the maximum estimate is not the action that has the minimum penalty probability, then the automaton pursues a wrong action. In this paper, we generalize the design of the Pursuit algorithms such that it pursues a set of actions. Specifically, these actions have higher reward estimates than the current chosen action.

Figure 1 presents a pictorial representation of the two Pursuit approaches of converging to an action. The first approach, adopted by the existing Pursuit Algorithms, such as CP_{RP}, CP_{RI}, DP_{RP}, DP_{RI}, always pursues the best-estimated action. The present approach, adopted by the Generalized Pursuit Algorithms which we present here, does not follow only the best action - it follows all the actions that are "better" than the current chosen action.

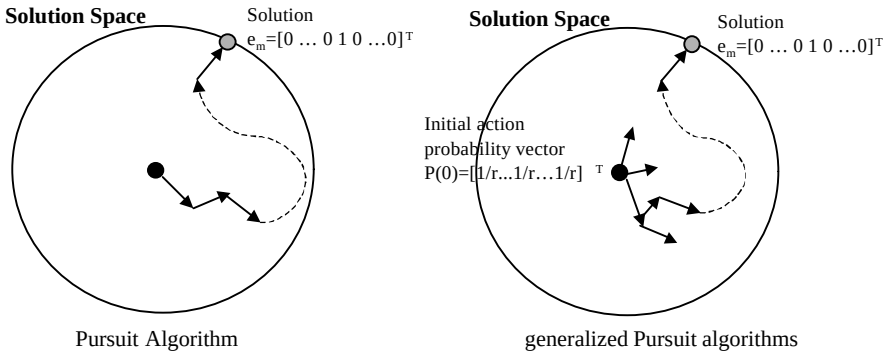

Figure 1: *Solution approach of the CP_{RP} Pursuit and Generalized Pursuit algorithms*

In a vectorial form, if action α_m is the action that has the highest reward estimate at time 't', the Pursuit Algorithms always pursue the vector $\mathbf{e}(t) = [0\ 0\ \dots\ 1\ 0 \dots 0\ ]^T$, where $e_m(t)=1$. In contrast, if α_i denotes the chosen action, the Generalized Pursuit algorithms pursues the vector **e**(t), where

$$\begin{aligned} e_j(t) &= \begin{cases} 0, & \text{if } \hat{d}_j(t) < \hat{d}_i(t) \\ 1, & \text{if } \hat{d}_j(t) \geq \hat{d}_i(t) \end{cases} \quad \text{for} \quad j \neq i \\ e_i(t) &= \begin{cases} 1, & \text{if } \hat{d}_i(t) = \max\{\hat{d}_j(t)\} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

Since this vector **e**(t) represents the direction towards which the probability vector moves, it is considered the *direction vector* of the Pursuit Algorithms.

We present two versions of Generalized Pursuit algorithms, followed by a comparative study of the performance of these algorithms with the existing Pursuit algorithms. The first algorithm introduced is the

Generalized Pursuit Algorithm (GPA). This algorithm moves the action probability vector "away" from the actions that have smaller reward estimates, but it does not guarantee that it increases the probability for all the actions with higher estimates than the chosen action.

Next, we present a pseudo-discretized Generalized Pursuit Algorithm (pDGPA). This algorithm follows the philosophy of a Generalized Pursuit Algorithm in the sense that it increases the action probability for all the actions with higher reward estimates than the current chosen action. Since the action probabilities are increased in discrete unequal steps, this algorithm is considered a *pseudo-discretized* Generalized Pursuit Algorithm.

III.1. Generalized Pursuit Algorithm

The Generalized Pursuit Algorithm (GPA) presented in this section, is an example of an algorithm that generalizes the Pursuit Algorithm CP_{RP} introduced by Thathachar and Sastry in [24]. It is a continuous estimator algorithm, which moves the probability vector towards a set of possible solutions in the probability space. Each possible solution is a unit vector in which the value '1' corresponds to an action that has a higher reward estimate than the chosen action.

The CP_{RP} algorithm increases the probability for the action that has the higher reward estimate, and decreases the action probability for all the other actions. To increase the probability for the best-estimated action and to also preserve **P**(t) a probability vector, the Thathachar and Sastry's Pursuit algorithm first decreases the probabilities of all actions:

$$p_j(t+1) = (1-\lambda)p_j(t), \qquad j = 1,\ldots,r \tag{2}$$

The remaining amount Δ that determines the sum of the action probabilities to be '1' is computed as :

$$\Delta = 1 - \sum_{j=1}^{r} p_j(t+1) = 1 - \sum_{j=1}^{r} (1-\lambda)p_j(t) = 1 - \sum_{j=1}^{r} p_j(t) + \lambda \sum_{j=1}^{r} p_j(t) = \lambda$$

In order to increase the probability of the best-estimated action, the CP_{RP} Pursuit algorithm adds the probability mass Δ to the probability of the best-estimated action:

$$p_m(t+1) = (1-\lambda)p_m(t) + \Delta = (1-\lambda)p_m(t) + \lambda, \qquad \text{where } \hat{d}_m = \max_{j=1,\ldots,r}\{\hat{d}_j(t)\} \tag{3}$$

In contrast to the CP_{RP} algorithm, the newly introduced GPA algorithm equally distributes the remaining amount Δ to all the actions that have higher estimates that the chosen action. If K(t) denotes the number of actions that have higher estimates than the chosen action at time 't', then the updating equations for the Generalized Pursuit Algorithm are expressed by the following equations:

$$
\begin{aligned}
p_j(t+1) &= (1-\lambda)p_j(t) + \frac{\lambda}{K(t)}, && \text{for all } j, \text{ such that } \hat{d}_j(t) > \hat{d}_i(t) \\
p_j(t+1) &= (1-\lambda)p_j(t), && \text{for all } j, \text{ such that } \hat{d}_j(t) \le \hat{d}_i(t) \\
p_i(t+1) &= 1 - \sum_{j \neq m} p_j(t).
\end{aligned} \tag{4}
$$

In vector form, the updating equations can be expressed as follows:

$$
\mathbf{P}(t+1) = (1-\lambda)\cdot \mathbf{P}(t) + \frac{\lambda}{K(t)} \cdot \mathbf{e}(t), \tag{5}
$$

where **e**(t) is the *direction vector* defined in (1).

Based on these equations, it can be seen that the GPA algorithm increases the probability for all the actions with higher reward estimates than the estimate of the chosen action, and satisfying the following inequality:

$$
p_j(t) < \frac{1}{K} \tag{6}
$$

Formally, the Generalized Pursuit Algorithm can be described as follows:

ALGORITHM GPA

Parameters

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

m index of the maximal component of $\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,..,r}\{\hat{d}_i(t)\}$

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

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

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking 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 K(t) represents the number of actions with higher estimates than the chosen action at time t, update **P**(t) according to the following equations:

$$
\begin{aligned}
p_j(t+1) &= p_j(t) - \lambda \cdot p_j(t) + \frac{\lambda}{K(t)} && (\forall j) \text{ such that } \hat{d}_j(t) > \hat{d}_i(t) \\
p_j(t+1) &= p_j(t) - \lambda \cdot p_j(t) && (\forall j) \text{ such that } \hat{d}_j(t) < \hat{d}_i(t) \\
p_i(t+1) &= 1 - \sum_{j \neq i} p_j(t+1)
\end{aligned} \tag{7}
$$

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

End Repeat
END ALGORITHM GPA

As in the case of the previous Pursuit algorithms, the convergence of the GPA is proven in two steps. First, we demonstrate that using a sufficiently small value for the learning parameter λ, all actions are chosen enough number of times such that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. These are formalized below.

Theorem 4: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the GPA algorithm, for all $\lambda \in (0, \lambda^*)$,

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

Proof: The proof of this theorem is found in the unabridged version of this paper. ♦ ♦ ♦

The second step in proving the convergence of the GPA consists of demonstrating that if the m^{th} action is rewarded more than any other action from time t_0 onward, then the action probability vector converges in probability to $\mathbf{e}_m$. This is shown in the following theorem.

Theorem 5: 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 $t \rightarrow \infty$.

Proof: : The proof of this theorem is also found in the unabridged version of this paper. ♦ ♦ ♦

Finally, the ε-optimal convergence result can be stated as follows:

Theorem 6: For the GPA 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 $\varepsilon \in (0, 1)$,

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

for all $t > t_0$. ♦ ♦ ♦

The simulation results regarding the performance of this algorithm are presented in the section IV.

III.2. Pseudo-Discretized Generalized Pursuit Algorithm

The Pseudo-Discretized Generalized Pursuit Algorithm, denoted pDGPA, is another algorithm that generalizes the concepts of the Pursuit algorithm by 'pursuing' all the actions that have higher estimates than the current chosen action.

At each iteration, the algorithm counts how many actions have higher estimates than the current chosen action. If K(t) denotes this number, the pDGPA algorithm increases the probability of all the actions with higher estimates with the amount $\Delta/K(t)$, and decreases the probabilities for all the other actions with the amount $\Delta/(r-K(t))$, where Δ is a resolution step, $\Delta=1/rN$ with N a resolution parameter.

Vectorially, the updating equations can be expressed as follows:

$$\mathbf{P}(t+1) = \mathbf{P}(t) + \frac{\Delta}{K(t)} \cdot \mathbf{e}(t) - \frac{\Delta}{r - K(t)} \cdot \left[\mathbf{u} - \mathbf{e}(t)\right] \qquad (8)$$

where $\mathbf{e}(t)$ is the direction vector defined in (Eq. 1) and $\mathbf{u}$ is the unit vector $u_j=1$, $j=1,2,\ldots,r$.

A detailed description of the algorithm is given below:

ALGORITHM pDGPA

Parameters

N resolution parameter

K(t) the number of actions with higher estimates than the current chosen action

Δ the smallest step size $\Delta = \frac{1}{rN}$

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

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

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking 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: Update $\mathbf{P}(t)$ according to the following equations:

$$p_j(t+1) = \min\{p_j(t) + \frac{\Delta}{K(t)}, 1\} \qquad (\forall j) \text{ such that } \hat{d}_j(t) > \hat{d}_i(t)$$

$$p_j(t+1) = \max\{p_j(t) - \frac{\Delta}{r - K(t)}, 0\} \qquad (\forall j) \text{ such that } \hat{d}_j(t) < \hat{d}_i(t) \qquad (9)$$

$$p_i(t+1) = 1 - \sum_{j \neq i} p_j(t+1)$$

Step 3: Same as in the GPA algorithm

End Repeat

END ALGORITHM pDGPA

It can be seen that the pDGPA always increases the probability of all the actions with higher estimates. To prove the convergence of this algorithm, we assert that the pDGPA possesses the moderation and monotone properties [16], and this is proved in the unabridged paper. The ε-optimal results follow.

Theorem 7: The pDGPA possesses the moderation property. ♦ ♦ ♦

Theorem 8: The pDGPA possesses the monotone property. ♦ ♦ ♦

IV. EXPERIMENTAL RESULTS

This section presents comparison of the performance of the newly introduced Generalized Pursuit Algorithms. In order to compare their relative performances, we performed simulations to accurately characterize their respective rates of convergence. The simulations were performed imposing the same restrictions and in the same benchmark environments as the simulations presented in [20]. 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 ($0<T\leq1$). 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.

The simulations were performed for different existing benchmark environments with ten actions, for which the threshold T was considered 0.999 and NE=750, the same values used in [20]. 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 [24]. 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. The results are presented in Table 1. For comparison, Table 2 presents the simulation results of the existing Pursuit algorithms.

Table 1: Performance of the Generalized Pursuit algorithms in ten-action environments for which exact convergence was required in 750 experiments.

Environment	GPA		pDGPA	
	λ	No. of Iterat.	N	No. of Iterat.
E_A	0.0127	948.03	24	633.64
E_B	0.0041	2759.02	52	1307.76

Note: 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

Table 2: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 750 experiments [20].

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

Since the GPA algorithm was designed as a generalization of the continuous reward-penalty Pursuit algorithm, a comparison between these two algorithms is presented. The results show that the GPA algorithm is 60% faster than the CP_{RP} algorithm in the E_A environment and 51% faster in the E_B environment. For example, in the E_A environment, the GPA converges in average in 948.03 iterations, and the CP_{RP} algorithm requires in average 2427 iterations for convergence, which shows an improvement of 60%. In the E_B environment, the CP_{RP} algorithm required on average 5685 number of iterations whereas the GPA algorithm required on average only 2759.02 iterations for convergence, being 51% faster than the CP_{RP} algorithm.

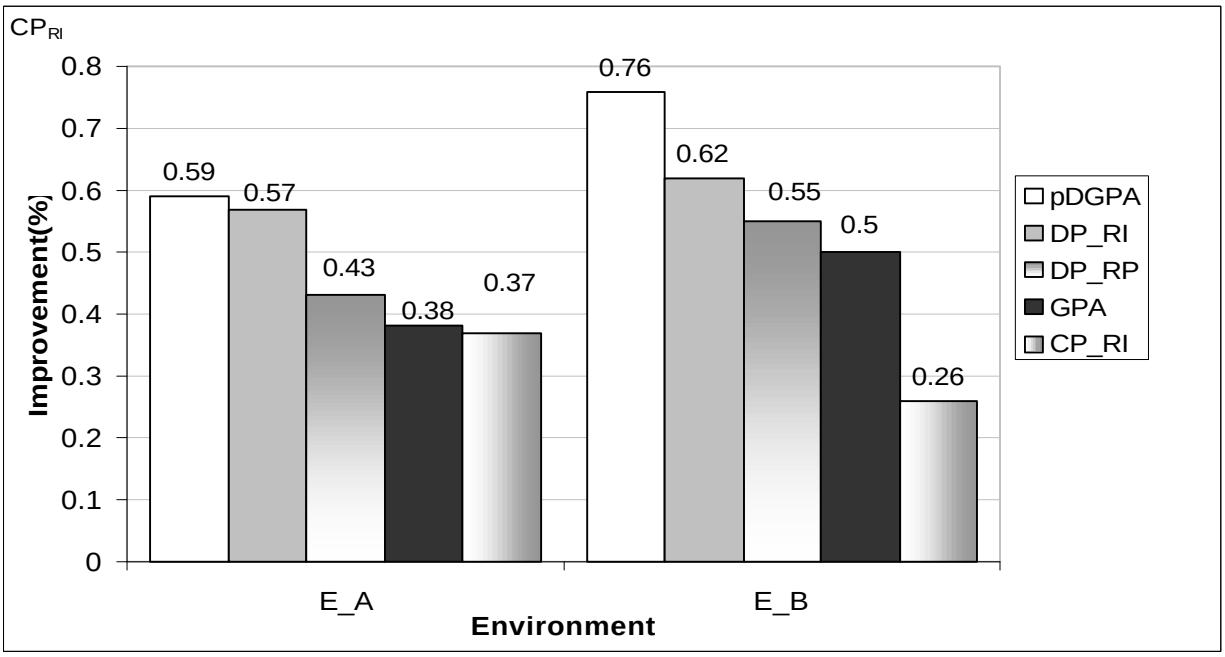

Figure 2: Performance of the Pursuit Algorithms relative to the CP_{RP} algorithm in ten-action environments for or which exact convergence was required in 750 experiments.

Similarly, the pDGPA is classified as a reward-penalty discretized Pursuit algorithm. If compared against the DP_{RP} algorithm, the pDGPA proves to be up to 59% faster. For example, in the E_B environment, the pDGPA algorithm converges in an average of 1307.76 iterations whereas the DP_{RP} algorithm requires 3230 iterations. Also, the pDGPA algorithm proves to be the fastest Pursuit algorithm, being up to 50% faster than the DP_{RI} algorithm.

Figure 2 presents the graphical representation of the relative performance of these algorithms to the CP_{RP} algorithm in benchmark ten-action environments.

Based on these experimental results and considering the number of iterations required to attain the same accuracy of convergence in ten-action benchmark environments, we can rank the six Pursuit algorithms as follows:

Best Algorithm:	pseudo-Discretized Generalized Pursuit Algorithm (pDGPA)	
2nd-best Algorithm:	Discretized Pursuit Reward-Inaction	(DP_{RI})
3nd-best Algorithm:	Generalized Pursuit Algorithm	(GPA)
4rd-best Algorithm:	Discretized Pursuit Reward-Penalty	(DP_{RP})
5rd-best Algorithm:	Continuous Pursuit Reward-Inaction	(CP_{RI})
6th-best Algorithm:	Continuous Pursuit Reward-Penalty	(CP_{RP})

V. CONCLUSION

In this paper, we introduced a generalization of the learning method of the Pursuit algorithms that *'pursues'* the set of actions that have higher estimates than the current chosen action. We have argued that this minimizes the probability of pursuing a wrong action. We presented two new generalized Pursuit algorithms that follow this learning approach, namely, the Generalized Pursuit Algorithm (GPA) and the pseudo-discretized Generalized Pursuit Algorithm (pDGPA), and also presented a quantitative comparison between these algorithms and the existing Pursuit algorithms. In the same environments, while the GPA algorithm proved to be the fastest continuous Pursuit algorithm, the pDGPA proved to be the fastest converging discretized Pursuit estimator algorithm. It is the fastest Pursuit estimator algorithm.

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 Mathematical 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] B.J. Oommen and M. Agache, "A Comparison of Continuous and Discretized Pursuit Learning Schemes", *Proceedings of the 1999 IEEE. International Conference on Syst. Man. Cybern*., October, 1999, Tokyo, Japan. pp. IV:1061-1067.
[21] R. Ramesh, *Learning Automata in Pattern Classification"*, M.E. Thesis, Indian Institute of Science, Bungalore, India, 1983.
[22] M. A. L. Thathachar and B. J. Oommen, "Discretized Reward-Inaction Learning Automata", *J. Cybern. Information Sci*., pp. 24-29, Spring 1979.
[23] 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.
[24] 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.
[25] M.L. Tsetlin, "On the Behavior of Finite Automata in Random Media", *Automat. Telemek.* (USSR), Vol. 22, pp. 1345-1354, Oct. 1961.
[26] M.L. Tsetlin, *Automaton Theory and the Modeling of Biological Systems*, New York: Academic, 1973.
[27] 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.