Generalized Pursuit Learning Schemes: New Families of Continuous and Discretized Learning Automata

Mariana Agache and B. John Oommen, *Senior Member, IEEE*

***Abstract*—The fastest learning automata (LA) algorithms currently available fall in the family of estimator algorithms introduced by Thathachar and Sastry [24]. The pioneering work of these authors was the pursuit algorithm, which *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, we present two new generalized pursuit algorithms (GPAs) and also present a quantitative comparison of their performance against the existing pursuit algorithms. Empirically, the algorithms proposed here are among the fastest reported LA to date.**

***Index Terms*—Estimator algorithms, learning automata (LA), pursuit algorithms.**

I. Introduction

THE GOAL of a learning automaton (LA) 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 fastest LA algorithms to date fall within the family of the so-called *estimator algorithms*. From this family, the pioneering work of Thathachar and Sastry [24] was the pursuit algorithm, which 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 present various generalizations of the above philosophy. Quite simply stated, we demonstrate that a pursuit scheme, which 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, thus leading to a faster converging scheme. Based on *this* fundamental premise, we devise new pursuit LA which, empirically, are probably the fastest and most accurate reported LA to date.

A. Fundamentals of Learning Automata

The functionality of the LA 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. Excellent books that survey the field are [2], [9], and [31].

The learning paradigm as modeled by LA has found applications in systems that possess incomplete knowledge about the environment in which they operate. A variety of applications[1] that use LA have been reported in the literature. They have been used in game playing [1]–[3], pattern recognition [10], [21], object partitioning [17], [18], parameter optimization [9], [28], [35], [54] and multi-objective analysis [43], telephony routing [11], [12], and priority assignments in a queuing system [7]. They have also been used in statistical decision making [9], [31], distribution approximation [40], natural language processing, modeling biological learning systems [26], string taxonomy [56], graph partitioning [57], distributed scheduling [39], network protocols (including conflict avoidance [44]) for LANs [36], photonic LANs [45], star networks [37], broadcast communication systems [38], dynamic channel allocation [38], tuning PID controllers [30], assigning capacities in prioritized networks [32], map learning [41], digital filter design [42], controlling client/server systems [46], adaptive signal processing [49], vehicle path control [50], and even the control of power systems [51] and vehicle suspension systems [52].

The beauty of incorporating LA in any particular application domain, is indeed, the elegance of the technology. Essentially, LA are utilized exactly as one would expect—by interacting with the "environment"—which is the system from which the LA learn. For example, in parameter optimization, the LA chooses a parameter, observes the effect of the parameter in the control loop, and then updates the parameter to optimize the objective function, where this updating is essentially achieved by the LA stochastic updating rule. The details of how this is achieved in the various application domains involves modeling the "actions," transforming the system's outputs so that they are perceived to be of a reward or penalty flavor. This is where the ingenuity of the researcher comes into the picture—this is often a thought-provoking task. Although we have interests in the application of LA, this paper is intended to be of a theoretical sort.

Manuscript received November 15, 2001; revised March 1, 2002. The work of B. J. Oommen was supported in part by the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper can be found in the Proceedings of the Fourth World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, FL, July 2000. This paper was recommended by Associate Editors M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis.

The authors are with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: magache@nortelnetworks.com; oommen@scs.carleton.ca).

Publisher Item Identifier S 1083-4419(02)06462-2.

[1]*The applications listed here (and the references listed in the bibliography) are quite comprehensive, but by no means complete*. Indeed, this relatively new field has been "exploding." It has recently been enhanced by a spectrum of applications in computer science and engineering—from areas as diverse as the design of data structures, to the implementation of automatic navigation methods.

Thus, our discussions on the applications of LA are necessarily brief.

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 *et al.* [25], [26] presented notable examples of this type of automata. Later, in [27], Vorontsova and Varshavskii introduced a class of stochastic automata known in literature as variable structure stochastic automata (VSSA). In the definition of a VSSA, the LA is completely defined by a set of actions (one of 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 [2], [4], [9] operates on a vector (called the *action 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

$$p_i(t) = \Pr[\alpha(t) = \alpha_i], \qquad i = 1, \ldots, r$$

and it satisfies

$$\sum_{i=1}^{r} p_i(t) = 1 \qquad \text{for all } \text{“}t\text{.”}$$

Note that the algorithm T: $[0, 1]^r \times A \times B \rightarrow [0, 1]^r$ is an updating scheme where $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, $2 \leq r < \infty$, is the set of output actions of the automaton, and B is the set of responses from the environment. Thus, the updating is such that

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

where $\beta(t)$ is the response that the LA receives from the environment. These terms will be formalized presently, but it is well acclaimed in the LA literature.

If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as an *absorbing algorithm*. Families of VSSA that possess absorbing barriers have been studied in [4], [8]–[10]. Ergodic VSSA have also been investigated [2], [9], [10], [13]. These VSSA converge in distribution and thus, the asymptotic distribution of the action probability vector has a value that is independent of the corresponding initial vector. Thus, while ergodic VSSA are suitable for nonstationary environment, automata with absorbing barriers are preferred in stationary environments.

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 nonlinear. Following the discretization concept, many of the continuous VSSA have been discretized; indeed, various discrete automata have been presented in the literature [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. Thus, in essence, within the family of estimator algorithms, the LA's updating rule is of the form

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

where $\mathbf{Q}(t)$ is the pair $\langle \mathbf{P}(t), \hat{\mathbf{d}}(t) \rangle$ and $\hat{\mathbf{d}}(t)$ is the vector of estimates of the reward probabilities $\mathbf{d}$.

A detailed catalogue of the estimator algorithms can be found in [5], [6], [16], [23], [24], [33], [34], and [47]. Historically, as mentioned earlier, these algorithms were pioneered by Thathachar and Sastry, who introduced the continuous versions. Oommen *et al.* [5], [16] discretized them, thus enhancing them both with regard to speed and accuracy. Papadimitriou [33], [34] introduced the concept of *stochastic estimator algorithms* and recommended the use of windows for the estimates. He also used the nonlinear discretized version in a brilliant hierarchical manner for dealing with scenarios with a large number of actions. The most recent work in this field is the work that we have done in [20], which will be explained in greater detail.

Most of the analysis has centered around the ε-optimality of the corresponding LA, although the LA described in [34] have been proven to only possess the property of absolute expediency. However, we commend the excellent work of Rajaraman and Sastry [47] which is the only known *finite time* analysis for any of the estimator algorithms.

B. Contribution of This Paper

As mentioned earlier, 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. *Our primary goal here is to minimize this probability of pursuing an erroneous action*. It is pertinent to mention that the effect of minimizing this probability of erroneous pursuit leads to both faster and more accurate schemes, as we shall see. Furthermore, such a pursuit also leads to a new class of *pseudodiscretized* pursuit automata that are distinct from all the previously reported discretized LA. These contributions will be clear in the subsequent sections.

We achieve this our goal by generalizing the design of the pursuit algorithm by permitting it to pursue a set of actions[2] instead of a single "current-best" action. Specifically, the set of actions pursued are those actions that have higher reward estimates than the current chosen action. In this paper, we introduce two new pursuit algorithms, a continuous version and a discretized version, that use such a generalized "pursuit" learning approach. Apart from proving their learning properties, we also demonstrate their superiority over traditional pursuit and estimator schemes by means of extensive simulation results. The new algorithms presented here are among the best reported, and we believe represent the state of the art.

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 by $\mathrm{CP_{RP}}$. Later, in 1990, Oommen and Lanctôt [16] introduced the first discretized pursuit estimator algorithm by presenting a discretized version, denoted by $\mathrm{DP_{RI}}$, that uses a reward–inaction learning paradigm. In [20], Oommen and Agache 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 ($\mathrm{CP_{RI}}$) and the discretized reward–penalty pursuit algorithm ($\mathrm{DP_{RP}}$). In the interest of brevity, we shall present in this paper only the $\mathrm{CP_{RP}}$ and the $\mathrm{DP_{RI}}$ algorithms, which are currently recognized as the benchmark pursuit schemes.

A. The Continuous Pursuit Reward–Penalty ($\mathrm{CP_{RP}}$) Algorithm

The pioneering pursuit algorithm, the continuous pursuit algorithm, was introduced by Thathachar and Sastry [24]. We present it here in all brevity. This algorithm uses a *reward–penalty* learning paradigm, meaning that it updates the probability vector $\mathbf{P}(t)$, if the environment rewards or penalizes the chosen action. For this reason, we shall refer to it as the continuous pursuit reward–penalty ($\mathrm{CP_{RP}}$) algorithm. The $\mathrm{CP_{RP}}$ 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 or 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. From a vector perspective, the probability updating rules can be expressed as follows:

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

[2]Thathachar and Sastry also introduced the class of estimator algorithms (referred to as the "TSE estimator algorithms" [24]), which were subsequently discretized by Lanctôt and Oommen [6]. These algorithms possess the property that they do not just pursue the "current best action" but update the action probability vector using a weighting function, which, in turn, is a function of the differences between the reward estimates. The intention, in this paper, is to design and develop algorithms which are more straightforward, and which directly (as the crow flies) "pursue" the set of "superior" actions *without invoking any auxiliary functions*.

where e_m is the unit vector $[0 \cdots 1 \cdots 0]^T$ with the position of unity representing the currently estimated "best" action, namely, the action with the maximal reward *estimate*. 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, direction given by the vector e_m.

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, namely, $\mathbf{W}(t)$ and $\mathbf{Z}(t)$, where $Z_i(t)$ is the number of times the ith 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 shown below.

ALGORITHM $\mathrm{CP_{RP}}$

Parameters

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

m index of the maximal component of the reward estimate vector

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

$\mathbf{e}_m$ unit r-vector with 1 in the mth coordinate.

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

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

Notation: $\beta(t) \in \{0, 1\}$ is the response from the environment

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

$$\mathbf{P}(t+1) = (1-\lambda)\mathbf{P}(t) + \lambda \mathbf{e}_m. \tag{3}$$

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

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

End Repeat

END ALGORITHM $\mathrm{CP_{RP}}$

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

$\mathbf{P}(t)$ in the direction of the action which has the highest reward estimate.

In [24], Thathachar and Sastry 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 the 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.

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

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

The detailed proof for this result can be found in [24]. ♦♦♦

The second stage of the proof of convergence of the $\mathrm{CP_{RP}}$ 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 mth component of the action probability vector converges with probability one, to unity.

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

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

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

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

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

Using the assumptions of the theorem, this quantity becomes

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

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

$$E[p_m(t+1) - p_m(t) | Q(t)] \underset{t \to \infty}{\longrightarrow} 0 \text{ with probability one.}$$

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

The final theorem that shows the ε-optimal convergence of the $\mathrm{CP_{RP}}$ algorithm can be stated as Theorem 3.

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

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

for all $t > t_0$, where $\delta \in (0, 1)$ is arbitrarily small.

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

B. The Discretized Pursuit Reward–Inaction ($\mathrm{DP_{RI}}$) Algorithm

In 1990, Oommen and Lanctôt [16] introduced a discretized version of the pursuit algorithm. This pursuit algorithm was based on the *reward–inaction* learning paradigm, meaning that it updates the action probability vector $\mathbf{P}(t)$ only if the environment rewards the chosen action. In the context of this paper, we shall refer to this algorithm as the *discretized pursuit reward–inaction ($\mathrm{DP_{RI}}$) 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 $\mathbf{P}(t)$ in discrete steps, whereas the continuous versions use a continuous function to update $\mathbf{P}(t)$.

In the $\mathrm{DP_{RI}}$ algorithm, when an action is rewarded, all the actions that do not correspond to the highest estimate are decreased by a step Δ, where $\Delta = 1/rN$, and N is a resolution parameter. In order to keep the sum of the components of the vector $\mathbf{P}(t)$ 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 consequently, it is of the *reward–inaction* paradigm. This, in principle, fully describes the algorithm, given formally below.

ALGORITHM $\mathrm{DP_{RI}}$

Parameters

m — index of the maximal component of the reward estimate vector $\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,\ldots,r} \left\{\hat{d}_i(t)\right\}$.

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

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

N — resolution parameter.

$\Delta = 1/rN$ — smallest step size.

Notation: $\beta(t) \in \{0, 1\}$ is the response from the environment

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 $\mathbf{P}(t)$. Let $\alpha(t) = \alpha_i$.

Step 2: Update $\mathbf{P}(t)$ according to the following equations. If $p_j(t) = 0)$ and $p_m(t) \neq 1)$ then

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

$$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1). \tag{5}$$

Else

$$p_j(t+1) = p_j(t) \qquad \text{for all } 1 \leq j \leq r.$$

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

End Repeat

END ALGORITHM $\mathrm{DP_{RI}}$

Oommen and Lanctôt proved that this algorithm satisfies both the properties of moderation and monotonicity [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 $\mathrm{CP_{RP}}$ 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 mth action is rewarded more than any action from time t_0 onward, then the action probability vector for the $\mathrm{DP_{RI}}$ 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, where $j \neq m$ and all $t \geq t_0$. Then there exists an integer N_0 such that for all resolution parameters $N > N_0$, $p_m(t) \to 1$ with probability one as $t \to \infty$.

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

$$E[p_m(t+1) - p_m(t)|Q(t)] \underset{t \to \infty}{\longrightarrow} 0.$$

Indeed, the authors of [16] showed that

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

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

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

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 one. This, in turn, implies that $c_t \to 0$ with probability one. Consequently, that $\sum_{j \neq m} \max(p_j(t) - \Delta, 0) \to 0$ with probability one. Hence, $p_m(t) \to 1$ with probability one. ♦♦♦

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 $\mathrm{DP_{RI}}$, for all learning parameters $N > N_0$ and all time $t > t_0$

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

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

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

C. More Recent Pursuit Algorithms

In [20], Oommen and Agache explored all pursuit algorithms that resulted from the combination of the continuous and discrete probability spaces with the reward–penalty and reward–inaction learning paradigms. They showed that 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 $\mathbf{DP_{RI}}$: discretized pursuit reward–inaction scheme
Paradigm: reward–inaction; **Probability Space**: discretized.

Algorithm $\mathbf{DP_{RP}}$: discretized pursuit reward–penalty scheme
Paradigm: reward–penalty; **Probability Space**: discretized.

Algorithm $\mathbf{CP_{RI}}$: continuous pursuit reward–inaction scheme
Paradigm: reward–inaction; **Probability Space**: continuous.

Algorithm $\mathbf{CP_{RP}}$: continuous pursuit reward–penalty scheme
Paradigm: reward–penalty; **Probability Space**: continuous.

Observe that of the above four, the algorithms $\mathrm{CP_{RP}}$ and $\mathrm{DP_{RI}}$ were already presented in the literature and were described earlier. The more recent pursuit algorithms ($\mathrm{CP_{RI}}$ and $\mathrm{DP_{RP}}$) are described in detail in [20]. The details of their design and their comparative properties are omitted here in the interest of brevity.

III. Generalized Pursuit Algorithms (GPAs)

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 $\mathbf{P}(t)$ moves toward 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.

Fig. 1 presents a pictorial representation of the two pursuit approaches that can be used to converge to an action. The first approach, adopted by the existing pursuit algorithms, such as $\mathrm{CP_{RP}}$, $\mathrm{CP_{RI}}$, $\mathrm{DP_{RP}}$, and $\mathrm{DP_{RI}}$, always pursues the best-estimated action. The present approach, adopted by the generalized pursuit algorithms (GPAs) that we present here, does not follow only the best action—it follows all the actions that are "better" than the current chosen action, i.e., the actions that have higher reward estimates than the chosen action.

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 $\mathbf{e}(t) = [0\ 0\ \cdots\ 1\ 0\ \cdots\ 0]^T$, where $e_m(t) = 1$. In contrast, if α_i denotes the chosen action, the GPAs pursue the vector $\mathbf{e}(t)$, where

$$e_j(t) = \begin{cases} 0, & \text{if } \hat{d}_j(t) \leq \hat{d}_i(t) \\ 1, & \text{if } \hat{d}_j(t) > \hat{d}_i(t) \end{cases} \qquad \text{for } j \neq i$$

$$e_i(t) = \begin{cases} 1, & \text{if } \hat{d}_i(t) = \max\left\{\hat{d}_j(t)\right\} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

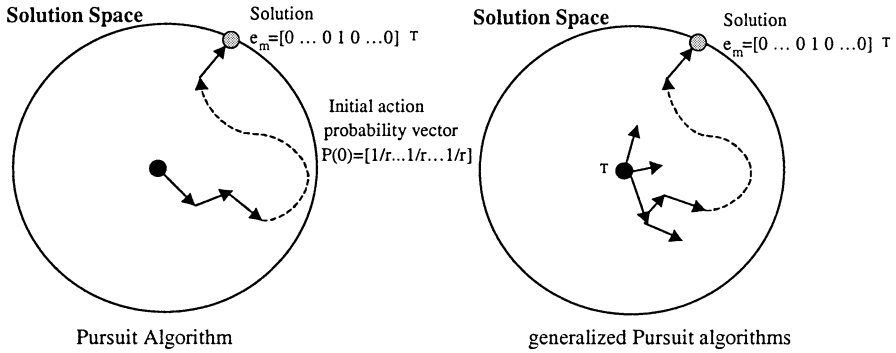


Fig. 1. Solution approach of the $\mathrm{CP_{RP}}$ pursuit and generalized pursuit algorithms.

Since this vector $\mathbf{e}(t)$ represents the direction toward which the probability vector moves, it is considered the *direction vector* of the pursuit algorithms.

In this paper, we present two versions of the GPAs, followed by a comparative study of the performance of these algorithms with the existing pursuit algorithms. The first algorithm introduced is the GPA. This algorithm moves the action probability vector "away" from the actions that have smaller reward estimates, but it ironically, does not guarantee that it increases the probability for all the actions with higher estimates than the chosen action.

In the subsequent section, we shall present a new algorithm. The latter follows the philosophy of a GPA in the sense that it increases the action probability for all the actions with higher reward estimates than the current chosen action.

A. The Generalized Pursuit Algorithm

The GPA presented in this section is an example of an algorithm that generalizes the pursuit algorithm $\mathrm{CP_{RP}}$ introduced by Thathachar and Sastry in [24]. It is a continuous estimator algorithm, which moves the probability vector toward 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 $\mathrm{CP_{RP}}$ algorithm increases the probability for the action that has the highest reward estimate, and decreases the action probability for all the other actions, as shown in the following updating equations:

$$\begin{aligned} p_m(t+1) &= (1-\lambda)p_m(t) + \lambda, \\ &\qquad \text{where } \hat{d}_m = \max_{j=1,\ldots,r}\left\{\hat{d}_j(t)\right\} \\ p_j(t+1) &= (1-\lambda)p_j(t), \qquad \text{for all } j \neq m. \end{aligned} \tag{7}$$

To increase the probability for the best-estimated action and to also preserve $\mathbf{P}(t)$ a probability vector, Thathachar and Sastry's pursuit algorithm first decreases the probabilities of all the actions:

$$p_j(t+1) = (1-\lambda)p_j(t), \qquad j = 1, \ldots, r. \tag{8}$$

The remaining amount Δ that determines the sum of the probabilities of all the actions to be "1" is computed as

$$\begin{aligned} \Delta &= 1 - \sum_{j=1}^{r} p_j(t+1) = 1 - \sum_{j=1}^{r} (1-\lambda)p_j(t) \\ &= 1 - \sum_{j=1}^{r} p_j(t) + \lambda \sum_{j=1}^{r} p_j(t) = \lambda. \end{aligned}$$

In order to increase the probability of the best-estimated action, the $\mathrm{CP_{RP}}$ pursuit algorithm adds the probability mass Δ to the probability of the best-estimated action

$$\begin{aligned} p_m(t+1) &= (1-\lambda)p_m(t) + \Delta = (1-\lambda)p_m(t) + \lambda \\ &\qquad \text{where } \hat{d}_m = \max_{j=1,\ldots,r}\left\{\hat{d}_j(t)\right\}. \end{aligned} \tag{9}$$

In contrast to the $\mathrm{CP_{RP}}$ algorithm, the newly introduced GPA algorithm equally distributes the remaining amount Δ to all the actions that have higher estimates than 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 GPA are expressed by the following equations:

$$\begin{aligned} p_j(t+1) &= p_j(t) - \lambda \cdot p_j(t) + \frac{\lambda}{K(t)} \\ &\qquad (\forall\, j) \text{ such that } \hat{d}_j(t) > \hat{d}_i(t),\ j \neq i. \\ p_j(t+1) &= p_j(t) - \lambda \cdot p_j(t) \\ &\qquad (\forall\, j) \text{ such that } \hat{d}_j(t) \leq \hat{d}_i(t),\ j \neq i. \\ p_i(t+1) &= 1 - \sum_{j \neq i} p_j(t+1). \end{aligned} \tag{10}$$

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

$$\mathbf{P}(t+1) = (1-\lambda) \cdot \mathbf{P}(t) + \frac{\lambda}{K(t)} \cdot \mathbf{e}(t) \tag{11}$$

where $\mathbf{e}(t)$ is the *direction vector* defined in (6).

Based on these equations, it can be seen that the GPA algorithm increases the probability for all the actions with $p_j(t) < 1/K(t)$, the reward estimates of which are higher than the reward estimate of the chosen action. Formally, the GPA can be described as shown below.

ALGORITHM GPA
Parameters
λ learning parameter, where $0 < \lambda < 1$.
m index of the maximal component of $\hat{\mathbf{d}}(t)$, $\hat{d}_m(t) = \max_{i=1,\ldots,r}\{\hat{d}_i(t)\}$.
$W_i(t)$, $Z_i(t)$: as in algorithms $\mathrm{CP_{RP}}$ and $\mathrm{DP_{RI}}$.
Notation: $\beta(t) \in \{0, 1\}$ is the response from the environment.
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: If $K(t)$ represents the number of actions with higher estimates than the chosen action at time t, update $\mathbf{P}(t)$ according to the following equations:

$$p_j(t+1) = p_j(t) - \lambda \cdot p_j(t) + \frac{\lambda}{K(t)} \quad (\forall j) \text{ such that } \hat{d}_j(t) > \hat{d}_i(t),\ j \neq i$$

$$p_j(t+1) = p_j(t) - \lambda \cdot p_j(t) \quad (\forall j) \text{ such that } \hat{d}_j(t) \le \hat{d}_i(t),\ j \neq i$$

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

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

End Repeat
END ALGORITHM GPA

As for 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. Formally, this is expressed as follows.

Theorem 6: 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 $\lambda \in (0, \lambda^*)$

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

Proof: The proof of this theorem is analogous to the proof of the corresponding result for the TSE algorithm [24]. We shall consider the same random variable Y_t^i as the number of times the ith action was chosen up to time t in any specific realization.

From (10), at any step t in the algorithm, if the action α_i is chosen, we have

$$p_j(t) = (1-\lambda) \cdot p_j(t-1) + \frac{\lambda}{K(t)-1} \quad \text{if } \hat{d}_j(t-1) > \hat{d}_i(t-1)$$

$$p_j(t) = (1-\lambda) \cdot p_j(t-1) \quad \text{if } \hat{d}_j(t-1) \le \hat{d}_i(t-1),\ j \neq i. \tag{12}$$

The probability of the chosen action $p_i(t)$ can either be $(1-\lambda) \cdot p_i(t-1)$, if there are other actions with better estimates than α_i, or, it can be $(1-\lambda) \cdot p_i(t-1) + \lambda$, if the chosen action has the maximal reward estimate. In both these situations, the following inequality is valid

$$p_i(t) \ge (1-\lambda) \cdot p_i(t-1).$$

Equation (10) shows that the following inequality is valid for all the actions:

$$p_j(t) \ge (1-\lambda) \cdot p_j(t-1), \qquad (\forall j) j = 1, \ldots, r \tag{13}$$

which implies that during any of the first t iterations of the algorithm

$$\Pr\{\alpha_i \text{ is chosen }\} \ge p_i(0) \cdot (1-\lambda)^t \quad \text{for any } i = 1, \ldots, r. \tag{14}$$

With the above clarified, the remainder of the proof is identical to the proof for the TSE algorithm and omitted (it can be found in [24]). ♦♦♦

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

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

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

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

Proof: We shall demonstrate that the sequence of random variables $\{p_m(t)\}_{t \ge t0}$ is a submartingale. The convergence in probability results then from the submartingale convergence theorem [9].

Consider

$$\Delta p_m(t) = E[p_m(t+1) - p_m(t) | \mathbf{Q}(t)] \tag{15}$$

where $\mathbf{Q}(t)$ is the state vector for the estimator algorithms which consists of $\mathbf{P}(t)$ and $\hat{\mathbf{d}}(t)$.

From the updating equations (8), and the assumptions of this theorem, $p_m(t+1)$ can be expressed as

$$p_m(t+1) = (1-\lambda) \cdot p_m(t) + \frac{\lambda}{K(t)} \quad \text{if } \alpha_j \text{ is chosen } j \neq m$$

$$p_m(t+1) = 1 - \sum_{j \neq m} (p_j(t) - \lambda \cdot p_j(t)) = p_m(t) + \lambda(1 - p_m(t)) \quad \text{if } \alpha_m \text{ is chosen.}$$

This implies that for all $t \ge t_0$, $\Delta p_m(t)$ can be calculated to be

$$\begin{aligned}\Delta p_m(t) &= \sum_{j \neq m} \left(\frac{\lambda}{K(t)} - \lambda \cdot p_m(t) \right) \cdot p_j(t) + [\lambda(1 - p_m(t))] \cdot p_m \\ &= \lambda \left(\frac{1}{K(t)} - p_m(t) \right) \cdot (1 - p_m(t)) + \lambda(1 - p_m(t)) \cdot p_m(t) \\ &= \frac{\lambda}{K(t)} (1 - p_m(t)) \ge 0. \end{aligned} \tag{16}$$

Hence, $p_m(t)$ is a submartingale. By the submartingale convergence theorem [9], $\{p_m(t)\}_{t \ge t0}$ converges as $t \to \infty$

$E[p_m(t+1) - p_m(t)|Q(t)] \to 0$ with probability one.

Hence, $p_m(t) \to 1$ with probability one, and the theorem is proven. ♦♦♦

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

Theorem 8: 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 proof for this theorem results as a logical consequence of the previous two theorems, Theorems 6 and 7. The simulation results regarding this algorithm are given in Section IV.

B. The Discretized Generalized Pursuit Algorithm (DGPA)

The discretized generalized pursuit algorithm (DGPA) 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. Strictly speaking, it is *pseudodiscretized* (as opposed to the various families of discretized schemes in the literature) because it moves the probability vector $\mathbf{P}(t)$ in discrete steps, but the steps do not have equal sizes, and are not "fixed" *a priori*.

At each iteration, the algorithm counts how many actions have higher estimates than the current chosen action. If $K(t)$ denotes this number, the DGPA 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.

From the perspective of vectors, the updating equations given in the formal algorithm below, are expressed as

$$\mathbf{P}(t+1) = \mathbf{P}(t) + \frac{\Delta}{K(t)} \cdot \mathbf{e}(t) - \frac{\Delta}{r - K(t)} \cdot [\mathbf{u} - \mathbf{e}(t)] \quad (17)$$

where $\mathbf{e}(t)$ is the direction vector defined in (6) and $\mathbf{u}$ is the unit vector in which $u_j = 1$, $j = 1, 2, \ldots, r$.

ALGORITHM DGPA

Parameters

N	resolution parameter.
$K(t)$	number of actions with higher estimates than the current chosen action.
Δ	smallest step size $\Delta = 1/rN$
$W_i(t)$, $Z_i(t)$:	as in algorithms $\mathrm{CP_{RP}}$ and $\mathrm{DP_{RI}}$.

Notation: $\beta(t) \in \{0, 1\}$ is the response from the environment.

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:

$$\begin{aligned} p_j(t+1) &= \min\left\{p_j(t) + \frac{\Delta}{K(t)}, 1\right\} \\ &\qquad (\forall j,\ j \neq i) \text{ such that } \hat{d}_j(t) > \hat{d}_i(t) \\ p_j(t+1) &= \max\left\{p_j(t) - \frac{\Delta}{r - K(t)}, 0\right\} \\ &\qquad (\forall j,\ j \neq i) \text{ such that } \hat{d}_j(t) < \hat{d}_i(t) \\ p_i(t+1) &= 1 - \sum_{j \neq i} p_j(t+1). \end{aligned} \quad (18)$$

Step 3: Same as in the GPA algorithm.

End Repeat

END ALGORITHM DGPA

In contrast to the GPA algorithm, the DGPA always increases the probability of all the actions with higher estimates.

To prove the convergence of this algorithm, we shall fist prove that the DGPA possesses the moderation property [16]. Then we prove that this algorithm also possesses the monotone property. As a consequence of these properties, the Markov chain $\{p_m(t)\}$ is a submartingale [16], and by the submartingale convergence theorem [9], the convergence of the algorithm is as follows.

Theorem 9: The DGPA possesses the moderation property.

Proof: To prove this property, we shall demonstrate that the value $1/rN$ bounds the magnitude by which any action probability can decrease at any iteration of the algorithm

$$p_j(t) - p_j(t+1) < \frac{1}{rN}.$$

From the updating equations in (17), the amount that a probability decreases is computed to be

$$p_j(t) - p_j(t+1) = \frac{\Delta}{r - K} = \frac{1}{rN} \cdot \frac{1}{r - K} < \frac{1}{rN}$$

and the result is proven. ♦♦♦

The second step in proving the convergence of the DGPA consists of demonstrating that the DGPA possesses the monotone property. This is shown in Theorem 10.

Theorem 10: The DGPA possesses the monotone property. Thus if there exists an index m and a time instant $t_0 < \infty$, such that

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

then there exists an integer N_0 such that for all $N > N_0$, $p_m(t) \to 1$ with probability one as $t \to \infty$.

Proof: The proof of 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

$$E[p_m(t+1) - p_m(t)|Q(t)] \underset{t \to \infty}{\longrightarrow} 0.$$

Consider

$$\Delta p_m(t) = E[p_m(t+1) - p_m(t)|\mathbf{Q}(t)]$$

where $\mathbf{Q}(t)$ is the state vector for the estimator algorithms.

From the updating equations in (17) and the assumptions of this theorem, $p_m(t+1)$ can be expressed as

$$p_m(t+1) = p_m(t) + \frac{\Delta}{K(t)} \qquad \text{if } \alpha_j \text{ is chosen, } j \neq m$$

$$p_m(t+1) = 1 - \sum_{j \neq m} \left(p_j(t) - \frac{\Delta}{r-1} \right)$$

$$= p_m(t) + \Delta \qquad \text{if } \alpha_m \text{ is chosen.}$$

This implies that for all $t \geq t_0$, $\Delta p_m(t)$ can be calculated to be

$$E[p_m(t+1) - p_m(t)|\mathbf{Q}(t)]$$

$$= \sum_{j \neq m} \frac{\Delta}{K(t)} \cdot p_j(t) + \Delta \cdot p_m(t)$$

$$= \frac{\Delta}{K(t)} (1 - p_m(t)) + \Delta \cdot p_m(t)$$

$$= \frac{\Delta}{K(t)} + \Delta \cdot p_m(t) \left(1 - \frac{1}{K(t)} \right) > 0.$$

Hence, $p_m(t)$ is a submartingale. By the submartingale convergence theorem [9], $\{p_m(t)\}_{t \geq t0}$ converges as $t \to \infty$, and thus

$$E[p_m(t+1) - p_m(t)|\mathbf{Q}(t)] \to 0 \text{ with probability one.}$$

Hence, $p_m(t) \to 1$ with probability one, and the theorem is proven. ♦♦♦

Since the DGPA possesses the moderation and monotony properties, it implies that the DGPA is ε-optimal [16] in all random environments.

IV. Experimental Results

This section presents a comparison[3] of the performance of the newly introduced GPAs with the pursuit algorithms reported to date. 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 [5], [6], [16], [20], and [24]. 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 $T(0 < T \leq 1)$. 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 always 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.

[3]The comparison here is only between the various families of pursuit algorithms. A more complete comparison between these algorithms and the families of estimator algorithms is not included here. Such a comparison is presently being compiled. The point is that because there are various parameters involved, a fair comparison can be made only by allowing the LA to "play on a level field."

TABLE I
Performance of the Generalized Pursuit Algorithms in Benchmark Ten-Action Environments for Which Exact Convergence was Required in 750 Experiments

Environment	GPA		DGPA	
	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

The simulations were performed for different existing benchmark environments with two and ten actions, for which the threshold T was set to be 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 $\mathrm{DP_{RI}}$ in [16], and to compare the performance of the $\mathrm{CP_{RP}}$ against other traditional VSSA in [24]. Furthermore, to keep the conditions identical, each estimator algorithm sampled all actions ten times each in order to initialize the estimate vector. These extra iterations are also included in the results presented in Table Is and II.

In a two-action environment, the GPA algorithm reduces in a degenerate manner, to the $\mathrm{CP_{RP}}$ pursuit algorithm, and the DGPA reduces to $\mathrm{DP_{RP}}$ algorithm. For this reason, simulations were performed only in ten-action benchmark environments and the results are presented in Table I. For comparison, Table II presents the simulation results of the existing pursuit algorithms.

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 $\mathrm{CP_{RP}}$ algorithm in the $\mathrm{E_A}$ environment and 51% faster in the $\mathrm{E_B}$ environment. For example, in the $\mathrm{E_A}$ environment, the GPA converges in average in 948.03 iterations, and the $\mathrm{CP_{RP}}$ algorithm requires, on average, 2427 iterations for convergence, which shows an improvement of 60%. In the $\mathrm{E_B}$ environment, the $\mathrm{CP_{RP}}$ 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 $\mathrm{CP_{RP}}$ algorithm.

Similarly, the DGPA is classified as a reward–penalty discretized pursuit algorithm. If compared against the $\mathrm{DP_{RP}}$ algorithm, the DGPA proves to be up to 59% faster. For example, in the $\mathrm{E_B}$ environment, the DGPA algorithm converges in an average of 1307.76 iterations whereas the $\mathrm{DP_{RP}}$ algorithm requires 3230 iterations. Also, the DGPA algorithm proves to be the fastest pursuit algorithm, being up to 50% faster than the $\mathrm{DP_{RI}}$ algorithm. For example, in the same environment, $\mathrm{E_B}$, the DGPA algorithm requires 1307.76 and the $\mathrm{DP_{RI}}$ algorithm requires 2693 iterations to converge.

TABLE II
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

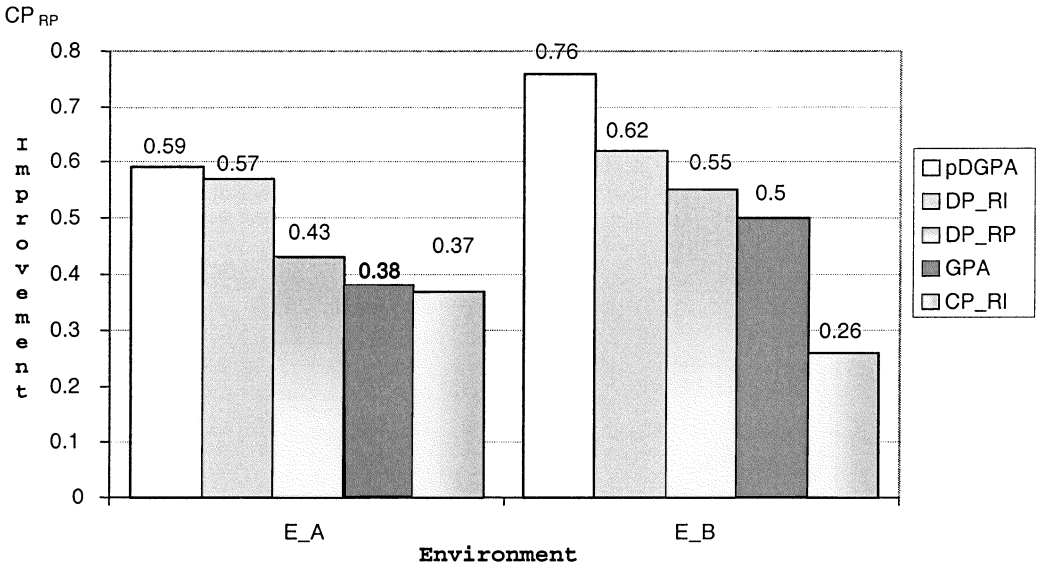


Fig. 2. Performance of the pursuit algorithms relative to the $\mathrm{CP_{RP}}$ algorithm in ten-action environments for which exact convergence was required in 750 experiments.

Fig. 2 graphically compares the relative performance of these algorithms to the $\mathrm{CP_{RP}}$ algorithm in the 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:
Discretized Generalized Pursuit Algorithm (DGPA)
2nd-best Algorithm:
Discretized Pursuit Reward–Inaction ($\mathrm{DP_{RI}}$)
3rd-best Algorithm:
Generalized Pursuit Algorithm (GPA)
4th-best Algorithm:
Discretized Pursuit Reward–Penalty ($\mathrm{DP_{RP}}$)
5th-best Algorithm:
Continuous Pursuit Reward–Inaction ($\mathrm{CP_{RI}}$)
6th-best Algorithm:
Continuous Pursuit Reward–Penalty ($\mathrm{CP_{RP}}$)

V. CONCLUSION

Over the last two decades, many new families of LA have emerged, with the class of estimator algorithms being among the fastest ones. Thathachar and Sastry [24] were the first to introduce the concept of pursuit algorithms. Various other continuous and discrete pursuit algorithms were introduced in [16], [20], but all these pursuit algorithms "pursue" the action that has the maximal reward estimate at any iteration.

In this paper, we introduced a generalization of the learning method of the pursuit algorithms that "*pursues*" the actions that have higher estimates than the current chosen action, and hence minimizing the probability of pursuing a wrong action. We presented two new GPAs that follow this learning approach, namely, the GPA and the DGPA. We have also presented a quantitative comparison between these algorithms and the existing pursuit algorithms.

Overall, the GPAs proved to be faster than the pursuit algorithms in environments with more than two actions. Specifically, in the same environments, the GPA algorithm proved to be the fastest continuous pursuit algorithm, the DGPA proved to be the fastest converging discretized pursuit estimator algorithm, and, indeed, the fastest pursuit estimator algorithm.

The question of studying multiple response automata [29] and parallel ensembles [48] of these new pursuit schemes, and of deriving finite time analysis for these new algorithms [47] remains open.

REFERENCES

[1] N. Baba, T. Soeda, and Y. Sawaragi, "An application of stochastic automata to the investment game," *Int. J. Syst. Sci.*, vol. 11, pp. 1447–1457, Dec. 1980.

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

[3] ——, "Two person decentralized team with incomplete information," *Appl. Math. Comput.*, vol. 8, pp. 51–78, 1981.

[4] S. Lakshmivarahan and M. A. L. Thathachar, "Absolutely expedient algorithms for stochastic automata," *IEEE Trans. Syst., 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. Statist., Carleton Univ., Ottawa, ON, Canada, 1989.

[6] J. K. Lanctôt and B. J. Oommen, "Discretized estimator learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1473–1483, Nov. 1992.

[7] M. R. Meybodi, "Learning automata and its application to priority assignment in a queuing system with unknown characteristic," Ph.D. dissertation, School Elect. Eng. Comput. Sci., Univ. Oklahoma, Norman.

[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] ——, "Learning automata—A survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-4, pp. 323–334, 1974.

[11] ——, "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. 18, pp. 451–458, May 1988.

[15] B. J. Oommen and E. R. Hansen, "The asymptotic optimality of discretized linear reward–inaction learning automata," *IEEE. Trans. Syst., Man, Cybern.*, vol. SMC-14, pp. 542–545, May 1984.

[16] B. J. Oommen and J. K. Lanctôt, "Discretized pursuit learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, pp. 931–938, July 1990.

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

[18] ——, "Stochastic automata solutions to the object partitioning problem," *Comput. J.*, vol. 35, pp. A105–A120, 1992.

[19] B. J. Oommen and M. A. L. Thathachar, "Multiaction learning automata possessing ergodicity of the mean," *Inf. Sci.*, vol. 35, pp. 183–198, June 1985.

[20] B. J. Oommen and M. Agache, "Continuous and discretized pursuit learning schemes: Various algorithms and their comparison," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 277–287, June 2001.

[21] R. Ramesh, "Learning automata in pattern classification," M.E. thesis, Indian Inst. Sci., Bangalore, India, 1983.

[22] M. A. L. Thathachar and B. J. Oommen, "Discretized reward–inaction learning automata," *J. Cybern. Inf. Sci.*, pp. 24–29, Spring 1979.

[23] M. A. L. Thathachar and P. S. Sastry, "A class of rapidly converging algorithms for learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 168–175, 1985.

[24] ——, "Estimator Algorithms for Learning Automata," in *Proc. Platinum Jubilee Conf. on Syst. Signal Processing*. Bangalore, India: Dept. Elect. Eng., Indian Inst. Sci., Dec. 1986.

[25] M. L. Tsetlin, "On the behavior of finite automata in random media," in *Proc. Automat. Telemek.*, vol. 22, USSR, Oct. 1961, pp. 1345–1354.

[26] ——, *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," in *Proc. Automat. Telemek.*, vol. 24, USSR, 1963, pp. 327–333.

[28] H. Beigy and M. R. Meybodi, "Adaptation of parameters of BP algorithm using learning automata," in *Proc. VI SBRN'00*.

[29] A. A. Economides, "Multiple response learning automata," *IEEE Trans. Syst., Man, Cybern. A*, vol. 26, pp. 153–156, Jan. 1996.

[30] M. N. Howell and M. C. Best, "On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata," *J. Control Eng. Prac.*, vol. 8, pp. 147–154, 1999.

[31] K. Najim and A. S. Poznyak, *Learning Automata: Theory and Applications*. New York: Pergamon, 1994.

[32] B. J. Oommen and T. D. Roberts, "Continuous learning automata solutions to the capacity assignment problem," *IEEE Trans. Comput.*, vol. 49, pp. 608–620, June 2000.

[33] G. I. Papadimitriou, "A new approach to the design of reinforcement schemes for learning automata: Stochastic estimator learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 6, pp. 649–654, 1994.

[34] ——, "Hierarchical discretized pursuit nonlinear learning automata with rapid convergence and high accuracy," *IEEE Trans. Knowl. Data Eng.*, vol. 6, pp. 654–659, 1994.

[35] A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*. New York: Springer-Verlag, 1997.

[36] G. I. Papadimitriou, P. Nicopolitidis, and A. S. Pomportsis, "Self-adaptive polling protocols for wireless LANs: A learning-automata-based approach," in *Proc. ICECS*, Malta, Sept. 2001.

[37] G. I. Papadimitriou and A. S. Pomportsis, "Self-adaptive TDMA protocols for WDM star networks: A learning-automata approach," *IEEE Photon. Technol. Lett.*, vol. 11, no. Oct., pp. 1322–1324, 1999.

[38] ——, "On the use of stochastic estimator learning automata for dynamic channel allocation in broadcast networks," in *Proc. IEEE/CEC 2000*, San Diego, CA, July 2000.

[39] F. Seredynski, "Distributed scheduling using simple learning machines," in *European Journal of Operational Research 107*. Amsterdam, The Netherlands: Elsevier, 1998, pp. 401–413.

[40] N. Abe and M. Warmuth, "On the computational complexity of approximating distributions by probabilistic automata," *Mach. Learn.*, vol. 9, pp. 205–260, 1998.

[41] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron, "Inferring finite automata with stochastic output functions and an application to map learning," *Mach. Learn.*, vol. 18, pp. 81–108, 1995.

[42] M. N. Howell and T. J. Gordon, "Countinous learning automata and adaptive digital filter design," in *Proc. UKACC Int. Conf. CONTROL* , 1998, IEE Conf. Publ. 455.

[43] K. S. Narendra and K. Parthasarathy, "Learning automata approach to hierarchical multiobjective analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 263–273, Jan. 1991.

[44] G. I. Papadimitriou and D. G. Maritsas, "Learning automata-based receiver conflict avoidance algorithms for WDM broadcast-and-select star networks," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 407–412, 1996.

[45] G. I. Papadimitriou and A. S. Pomportsis, "Learning-automata-based MAC protocols for photonic LANs," *IEEE Photon. Technol. Lett.*, vol. 14, p. 481, 2000.

[46] G. I. Papadimitriou, A. L. Vakali, and A. S. Pomportsis, "Designing a learning-automata-based controller for client/server systems: A methotodology," in *Proc. 12th IEEE ICTAI*, Vancouver, BC, Canada, Nov. 13–15, 2000.

[47] K. Rajaraman and P. S. Sastry, "Finite time analysis of the pursuit algorithm for learning automata," *IEEE Trans. Syst., Man, Cybern. A*, vol. 26, pp. 590–598, July 1996.

[48] M. A. L. Thathachar and M. T. Arvind, "Parallel algorithms for modules of learning automata," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, pp. 24–33, Jan. 1998.

[49] C. K. K. Tang and P. Mars, "Games of stochastic learning automata and adaptive signal processing," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 851–856, Aug. 1993.

[50] C. Unsal, P. Kachroo, and J. S. Bay, "Multiple stochastic learning automata for vehicle path control in an automated highway system," *IEEE Trans. Syst., Man, Cybern. A*, vol. 29, pp. 120–128, Jan. 1999.

[51] Q. H. Wu, "Learning coordinated control of power systems using interconnected learning automata," *Int. J. Electr. Power Energy Syst.*, vol. 17, no. 2, pp. 91–99, 1995.

[52] T. Gordon, C. Marsh, and Q. H. Wu, "Stochastic optimal control of vehicle suspension systems using learning automata," *J. Syst. Contr. Eng.*, pt. I, vol. 207, no. 13, pp. 143–152, 1993.

[53] F. Seredynski, J. P. Kitajima, and B. Plateau, "Simulation of learning automata networks on distributed computers," in *Proc. Eur. Simulation Symp.: Simulation and AI in Computer-Aided Techniques*, Dresden, Germany, Nov. 1992.

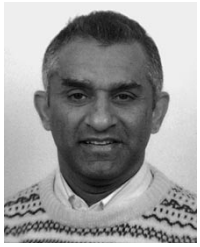
[54] A. S. Poznyak, K. Najim, and M. Chtourou, "Learning automata with continuous inputs and their application for multimodal functions optimization," *Int. J. Syst. Sci.*, vol. 27, no. 1, pp. 87–96, 1996.

[55] G. I. Papadimitriou and A. S. Pomportsis, "Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic," *IEEE Commun. Lett.*, vol. 4, pp. 107–109, 2000.

[56] B. J. Oommen and T. De St. Croix, "String taxonomy using learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 27, pp. 354–365, Apr. 1997.
[57] ——, "Graph partitioning using learning automata," *IEEE Trans. Comput.*, vol. 45, pp. 195–208, 1995.
[58] R. Simha and J. F. Kurose, "Relative reward strength algorithms for learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 388–398, 1989.

Mariana Agache received the B.S. degree in mathematics and computer science from the Al. I. Cuza University, Iasi, Romania, in 1992, and the M.C.S. degree in computer science from Carleton University, Ottawa, ON, Canada, in 2000.

Her research interests include learning machines and automata, pattern recognition, routing, and load balancing algorithms for data networks. She joined Nortel Networks in 1997, working in the ATM and MPLS Research and Development Group, Ottawa. Since 2000, she has been an ATM and MPLS Engineering Consultant for Nortel Networks in Europe.



B. John Oommen (S'79–M'83–SM'88) was born in Coonoor, India, on September 9, 1953. He received the B.Tech. degree from the Indian Institute of Technology, Madras, in 1975, the M.E. degree from the Indian Institute of Science, Bangalore, in 1977, and the M.S. and Ph.D. degrees from Purdue University, West Lafayette, IN, in 1979 and 1982, respectively.

He joined the School of Computer Science at Carleton University, Ottawa, ON, Canada, in the 1981–1982 academic year. He is still at Carleton and holds the rank of a Full Professor. His research interests include automata learning, adaptive data structures, statistical and syntactic pattern recognition, stochastic algorithms and partitioning algorithms. He is the author of more than 185 refereed journal and conference publication.

Dr. Oommen is on the editorial board of IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, and *Pattern Recognition.*