# Continuous and Discretized Pursuit Learning Schemes: Various Algorithms and Their Comparison

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

***Abstract*—A learning automaton (LA) 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 (LAs) have been proposed, with the class of estimator algorithms being among the fastest ones. Thathachar and Sastry, through the pursuit algorithm, introduced the concept of learning algorithms that pursue the current optimal action, following a *reward–penalty* learning philosophy. Later, Oommen and Lanctôt extended the pursuit algorithm into the discretized world by presenting the discretized pursuit algorithm, based on a *reward–inaction* learning philosophy. In this paper we argue that the reward–penalty and reward–inaction learning paradigms in conjunction with the continuous and discrete models of computation, lead to four versions of pursuit learning automata. We contend that a scheme that merges the pursuit concept with the most recent response of the environment, permits the algorithm to utilize the LAs *long-term* and *short-term* perspectives of the environment. In this paper, we present all four resultant pursuit algorithms, prove the $\epsilon$-optimality of the newly introduced algorithms, and present a quantitative comparison between them.**

***Index Terms*—Adaptive learning, estimator-based algorithms, learning automata.**

## 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 stochastic learning problem, Tsetlin, a Russian mathematician, created a new model of computer learning in 1961, now known as a *learning automaton* (LA). The goal of such an automaton is to determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded. The functionality of the learning automaton can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with the environment. During a cycle, the automaton chooses an action, which triggers a response from the environment, a response that can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. By learning to choose the optimal action, the automaton adapts itself to the environment.

Manuscript received July 23, 1999; revised February 3, 2001. A preliminary version of this paper can be found in the Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics, October 1999, Tokyo, Japan. The work of B. J. Oommen was supported in part by the Natural Sciences and Engineering Research Council of Canada. This paper was recommended by Associate Editor S. Lakshmivarahan.

B. J. Oommen is with the School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6 Canada (e-mail: oommen@scs.carleton.ca).

M. Agache is with Nortel Networks, Ottawa, ON K1P 6A9 Canada.

Publisher Item Identifier S 1083-4419(01)05216-5.

Learning automata (LAs) have found applications in systems that posses incomplete knowledge about the environment in which they operate, such as game playing [1]–[3], pattern recognition [10], [20], 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 LAs and their applications have been reviewed by Lakshmivarahan [2], and by Narendra and Thathachar [9]. Consequently, in this paper only a brief classification will be presented.

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

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

where $p_i(t)$ $(i = 1, \cdots, r)$ is the probability that the automaton will select the action $\alpha_i$ at the time $t$

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

$$\sum_{i=1}^{r} p_i(t) = 1, \quad \text{for all } t.$$

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

*Definition 1:* The VSSA is a 4-tuple $\langle A, B, T, P \rangle$, where $A$ is the set of actions, $B$ is the set of inputs of the automaton (the set of outputs of the environment), and $T : [0,1]^r \times A \times B \to [0,1]^r$ is an updating scheme such that

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

where

$\mathbf{P}(t)$ action probability vector;

$\alpha(t)$ action chosen at time $t$;

$\beta(t)$ response it has obtained.

In general, FSSA and VSSA can be analyzed using the theory of Markov chains. For the VSSA, if the mapping $T$ is independent of time, the probability $\mathbf{P}(t+1)$ is determined completely by $\mathbf{P}(t)$, which implies that $\{\mathbf{P}(t)\}_{t\geq 0}$ is a discrete-homogenous Markov process. From this perspective, different mappings $T$ can identify different types of learning algorithms. If the mapping $T$ is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as an *absorbing algorithm*. Families of VSSA that posses absorbing barriers have been studied in [4], [8], and [10]. Ergodic VSSA have also been investigated [2], [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. Because of this independence property, the ergodic VSSA are suitable for nonstationary environments. In stationary environments, automata with absorbing barriers may be preferred because they can be made to converge to the optimal action with a probability as close to unity as desired.

In practice, the relatively slow rate of convergence of these algorithms constituted a limiting factor in their applicability. In order to increase their speed of convergence, the concept of discretizing the probability space was introduced in [21]. This concept is implemented by restricting the probability of choosing an action to a finite number of values in the interval [0, 1]. If the values allowed are equally spaced in this interval, the discretization is said to be linear, otherwise, the discretization is called 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 attempting to design faster converging learning algorithms, Thathachar and Sastry [22] opened another avenue by introducing a new class of algorithms, called "estimator" algorithms. The main feature of these algorithms is that they maintain running estimates for the reward probability of each possible action, and use them in the probability updating equations. Typically, in the first step of the functional (reinforcement) 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. However, as we clarify later, estimator-type algorithms in the purest form make changes on the action probability vector based *only* on the running estimates of the reward probabilities—they ignore the feedback received from the environment, except to update the reward estimates themselves. A detailed description of the estimator algorithms can be found in [5], [6], [16], [22], and [23].

### A. Contribution of this Paper

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

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

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

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

The reader should observe that if the *long-term* properties of the environment are ignored and only the most recent responses are considered, the algorithm reduces to a traditional VSSA scheme, for example a reward–penalty or a reward–inaction scheme. Conversely, if the *short-term* responses of the environment are ignored, the algorithm reduces to the pursuit algorithm introduced in [23].

Historically, the pursuit algorithm presented by Thathachar and Sastry in [23] considered only the *long-term* estimates in the probability updating rules. Later, Oommen and Lanctôt [16] presented a discretized version of a pursuit algorithm which considered both the short-term and the long-term perspectives and embraced the *reward–inaction* learning paradigm. In this paper, we shall present these two pursuit algorithms and new versions of pursuit algorithms, which basically emerge from the

combination of these learning "philosophies" and paradigms. Apart from proving the $\varepsilon$-optimality of the newly introduced algorithms, we also submit a comprehensive quantitative comparison between the various algorithms.

## II. Pursuit Algorithms

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

### A. Continuous Pursuit Reward–Penalty ($CP_{R\Gamma}$) Algorithm

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

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

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

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

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

**ALGORITHM $CP_{R\Gamma}$**

**Parameters**

$\lambda$ speed of learning parameter where $0 < \lambda < 1$;

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

$\mathbf{e}_m$ unit $r$-vector with 1 in the $m$th coordinate;

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

$Z_i(t)$ 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 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 $\alpha_m$ 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

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

**End Repeat**

**END ALGORITHM $CP_{R\Gamma}$**

The $CP_{R\Gamma}$ algorithm is similar in design to the $L_{R\Gamma}$ 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 $L_{R\Gamma}$ 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 $CP_{R\Gamma}$ algorithm moves $\mathbf{P}(t)$ in the direction of the action which has the highest reward estimate.

Thathachar and Sastry [23] proved that this algorithm is $\varepsilon$-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 $\lambda$, all actions are chosen enough number of times so that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. This is stated in Theorem 1 below.

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

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

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

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

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

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

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

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

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

Using the assumptions of the theorem, this quantity becomes

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

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

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

This is equivalent to

$$\Delta p_m(t) = \lambda(1 - p_m(t)) \underset{t\to\infty}{\longrightarrow} 0 \text{ with probability 1}$$

and implies that $p_m(t) \to 1$ with probability 1.

The final theorem that shows the $\varepsilon$-optimal convergence of the $\text{CP}_{\text{RP}}$ algorithm can be stated as the following.

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

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

for all $t > t_0$.

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

### B. Discretized Pursuit Reward-Inaction (DPRI) Algorithm

In 1990, Oommen and Lanctôt introduced [16] a discretized version of a pursuit algorithm. Apart from the long-term perspective of the environment (recorded in the estimates of the reward probabilities), it also utilized the short-term perspective of the environment that was modeled in terms of the most recent environment response. This pursuit algorithm was based on the *reward–inaction* learning paradigm. In the context of this paper, we shall refer to this algorithm as the discretized pursuit reward–inaction ($\text{DP}_{\text{RI}}$) 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 $\text{DP}_{\text{RI}}$ algorithm, when an action is rewarded, all the actions that do not correspond to the highest estimate are decreased by a step $\Delta$, 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 $\Delta$. 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 $\text{DP}_{\text{RI}}$**

**Parameters**

| | |
|---|---|
| $m$ | index of the maximal component of the reward estimate vector $\hat{\mathbf{d}}(t), \hat{d}_m(t) = \max_{i=1,\cdots,r} \hat{d}_i(t)\}$; |
| $W_i(t)$ | number of times the $i$th action has been rewarded up to time $t$, for $1 \leq i \leq r$; |
| $Z_i(t)$ | number of times the $i$th action has been chosen up to time $t$, for $1 \leq i \leq r$; |
| $N$ | resolution parameter; |
| $\Delta = 1/rN$ | smallest step size. |

**Method**

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

**Initialize** $\hat{\mathbf{d}}(t)$ by choosing each action a 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** $\beta(t) = 0$ and $p_m(t) \neq 1$ **Then**

$$p_j(t+1) = \max_{j \neq m}\{p_j(t) - \Delta, 0\}$$
$$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1) \tag{5}$$

**Else**

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

Step 3) Update $\hat{\mathbf{d}}(t)$ exactly as in the $\text{CP}_{\text{RP}}$ Algorithm

**End Repeat**

**END ALGORITHM $\text{DP}_{\text{RI}}$**

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 $\varepsilon$-optimal in every stationary random environment.

The proof of the convergence of this algorithm follows the same trend as the proof for the $\text{Pursuit}_{RP}$ 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 $m$th action is rewarded more than any action from time $t_0$ onward then the action probability vector for the $\text{DP}_{\text{RI}}$ will converge to the unit vector $\mathbf{e}_m$. These results are stated below.

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

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

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

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

*Theorem 5:* For each action $\alpha_i$, 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 the Theorem 1, and can be found in [16].

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

## III. New Algorithms

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

**Algorithm DP:**$_{\mathrm{RI}}$ Discretized pursuit reward-inaction scheme.
**Paradigm:** Reward-inaction; **Probability Space**: Discretized.

**Algorithm DP:**$_{\mathrm{RP}}$ Discretized pursuit reward-penalty scheme.
**Paradigm:** Reward-penalty; **Probability Space:** Discretized.

**Algorithm CP:**$_{\mathrm{RI}}$ Continuous pursuit reward-inaction scheme.
**Paradigm:** Reward-inaction; **Probability Space:** Continuous.

**Algorithm CR$_{\mathbf{RP}}$**: Continuous pursuit reward-penalty scheme.
**Paradigm:** Reward-penalty; **Probability Space:** Continuous.

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

### A. Continuous Pursuit Reward-Inaction ($\mathrm{CP_{RI}}$) Algorithm

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

**ALGORITHM CP$_{\mathbf{RI}}$**
**Parameters**
$\lambda, m, \mathbf{e}_m, W_i(t), Z_i(t)$: Same as in the $\mathrm{CP_{RP}}$ algorithm.
**Method**
**Initialize** $p_i(t) = 1/r$, for $1 \leq i \leq r$
**Initialize** $\hat{\mathbf{d}}(t)$ by choosing each action a 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 $\alpha_m$ is the action with the current highest reward estimate, update $\mathbf{P}(t)$ as
**If** $R(t) = 0$ **Then**

$$\mathbf{P(t+1)} = \mathbf{(1-\lambda)P(t)} + \boldsymbol{\lambda}\mathbf{e}_m \tag{6}$$

**Else** $\mathbf{P}(t+1) = \mathbf{P}(t)$
Step 3) Update $\hat{\mathbf{d}}(t)$ exactly as in the $\mathrm{CP_{RP}}$ Algorithm.
**End Repeat**
**END ALGORITHM CP$_{\mathbf{RI}}$**

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

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

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

*Proof:* Let us define the random variable $Y_t^i$ as the number of times the $i$th action is chosen up to time $t$ in any specific realization. From the updating equation (6), at any step $t$ in the algorithm, we have

$$p_i(t) \geq p_i(t-1) \cdot (1 - \lambda) \tag{7}$$

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

$$\Pr\{\alpha_i \text{ is chosen}\} \geq p_i(0) \cdot (1-\lambda)^t. \tag{8}$$

With the above clarified, the reminder of the proof is identical to the proof of the $\text{CP}_{\text{RP}}$ algorithm and is omitted here to avoid repitition.

Analogous to the other Pursuit algorithms, it can be shown (Theorem 7) that if there is an action $\alpha_m$, for which the reward estimate remains maximal after a finite number of iterations, then the $m$th component of the action probability vector converges (w.p.i) to 1. Finally, Theorem 8 expresses the $\varepsilon$-optimality convergence for the $\text{CP}_{\text{RI}}$ algorithm, and can be directly deducted from the two previous results.

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

$$\hat{d}_m(t) > \hat{d}_j(t), (\forall j) j \neq m, (\forall t) t > t_0. \tag{9}$$

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

*Proof:* To prove this result, we shall demonstrate that the sequence of random variables $\{p_m(t)\}_{t \geq t0}$ is a submartingale. The strong convergence result will follow from the submartingale convergence theorem [9].

Based on the assumptions of the theorem, $p_m(t+1)$ can be expressed as

$$p_m(t+1) = p_m(t) + \lambda \cdot (1 - p_m(t)), \quad \text{if } \beta(t) = 0 \text{ (i.e., with probability } d_m) \tag{10}$$

$$p_m(t+1) = p_m(t), \quad \text{if } \beta(t) = 1 \text{ (i.e., with probability } 1 - d_m) \tag{11}$$

where $d_m$ is defined as $1 - c_m$, and $c_m$ is the penalty probability for the action $\alpha_m$.

To prove that $\{p_m(t)\}$ is a submartingale, we first prove that $\sup_{t \geq 0} E[|p_m(t)|] < \infty$.

$$\begin{aligned} E[p_m(t+1)|Q(t), p_m(t) \neq 1] &= d_m(t) \cdot (p_m(t) + \lambda \cdot (1 - p_m(t))) \\ &\quad + (1 - d_m(t)) \cdot p_m(t) \\ &= \lambda d_m(t) + p_m(t) \cdot (1 - \lambda d_m(t)). \end{aligned}$$

Since every term on the right-hand side of the equation is smaller than unity, it implies that $\sup_{t \geq 0} E[|p_m(t)|] < \infty$.

Based on (10) and (11), the quantity

$$\Delta p_m(t) = E[p_m(t+1) - p_m(t)|Q(t)] \tag{12}$$

becomes

$$\Delta p_m(t) = d_m \cdot \lambda \cdot (1 - p_m(t)) \geq 0, \quad \text{for all } t \geq t_0 \tag{13}$$

implying that $\{p_m(t)\}$ is a submartingale.

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

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

This is equivalent to

$$\Delta p_m(t) = d_m \cdot \lambda \cdot (1 - p_m(t)) \underset{t \to \infty}{\longrightarrow} 0 \text{ with probability 1}$$

which, in turn, means that $p_m(t) \to 1$ with probability 1, and the theorem is proven.

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

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

for all $t > t_0$.

*Proof:* The proof of this theorem is almost identical with the proof of the same theorem for the $\text{CP}_{\text{RP}}$ algorithm presented in [23]. We shall outline it below for the newly introduced $\text{CP}_{\text{RI}}$ algorithm.

Let $h$ be the difference between the two highest reward probabilities. By assumption $d_m$ is unique, and so there exists an $h > 0$, such that $d_m - h \leq d_i$ for all $i \neq m$. By the weak law of large numbers we know that if $\{X_j\}_{j=1 \cdots t}$ is a sequence of independent and identically distributed random variables, each having finite mean, $\mu$, then for any $\varepsilon > 0$

$$\Pr\left\{\left|\frac{X_1 + X_2 + \cdots + X_t}{t} - \mu\right| > \varepsilon\right\} \to 0 \text{ as } t \to \infty.$$

Let $X_t$ be the indicator function such that

$$X_t = 1, \quad \text{if } \alpha_i \text{ was rewarded the } t\text{th time } \alpha_i \text{ was chosen}$$
$$X_t = 0, \quad \text{if } \alpha_i \text{ was penalized the } t\text{th time } \alpha_i \text{ was chosen}$$

and let $Y_t^i$ be defined as in Theorem 6.

Hence, by the weak law of large numbers, we have the result that for a given $\delta > 0$ there exists an $M_i < \infty$, such that if $\alpha_i$ is chosen at least $M_i$ times

$$\Pr\left\{|\hat{d}_i(t) - d_i| < \frac{h}{2}\right\} > 1 - \delta.$$

Let $M = \max_{1 \leq i \leq r}\{M_i\}$. Since $h$ is strictly positive, it implies that for all $i \neq m$ and for all $t$, if $\min_{1 \leq i \leq r}\{Y_t^i\} > M$ then

$$\Pr\left\{|\hat{d}_i(t) - d_i| < \frac{h}{2}\right\} > 1 - \delta.$$

By Theorem 6, we know that we can define $t_0$ and $\lambda^*$ such that for all $i$, all $t > t_0$ and all $\lambda \in (0, \lambda^*)$

$$\Pr\{Y_t^i > M\} \geq 1 - \delta.$$

Thus, if all actions are chosen at least $M$ times, each of the $\hat{d}_i$ will be in a $h/2$ neighborhood of $d_i$ with an arbitrarily large probability.

However, we know that

$$d_m - h/2 > d_m - h \quad \text{because } h > 0$$
$$d_m - h/2 \geq d_i \quad \forall i \neq m.$$

Since the probability measure is continuous, and since for all events $\Pr\{A\} \geq \Pr\{AB\}$, we have

$$\lim_{t\to\infty} \Pr\{A\} \geq \lim_{t\to\infty} \Pr\{A|B\} \lim_{t\to\infty} \Pr\{B\}.$$

Let $A$ and $B$ be the following events:

$$A \equiv |p_m - 1| < \varepsilon$$

and

$$B \equiv \max_i\{|\hat{d}_i(t) - d_i|\} < h/2.$$

Then we have

$$\lim_{t\to\infty} \Pr\{|p_m(t) - 1| < \varepsilon\}$$
$$\geq \lim_{t\to\infty} \Pr\{|p_m(t) - 1| < \varepsilon | \max_i\{|\hat{d}_i(t) - d_i| < h/2\}$$
$$\cdot \lim_{t\to\infty} \Pr\{\max_i\{|\hat{d}_i(t) - d_i|\} < h/2\}.$$

Because the first term on the right-hand side goes to 1 based on Theorem 7 and the second term goes to $1-\delta$ by Theorem 6, we have

$$\lim_{t\to\infty} \Pr\{|p_m(t) - 1| < \varepsilon\} \geq 1 - \delta \quad \text{for all } \lambda \in (0, \lambda^*)$$

and the theorem is proven.

### *B. Discretized Pursuit Reward-Penalty ($DP_{RP}$) Algorithm*

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

**ALGORITHM $DP_{RP}$**

**Parameters**

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

**Method**

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

**Initialize** $\hat{\mathbf{d}}(t)$ by choosing each action a 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** $p_m(t) \neq 1$ **Then**

$$p_j(t+1) = \max_{j\neq m}\{p_j(t) - \Delta, 0\}$$
$$p_m(t+1) = 1 - \sum_{j\neq m} p_j(t+1). \tag{14}$$

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

**End Repeat**

**END ALGORITHM $DP_{RP}$**

Similar to the $DP_{RI}$ algorithm, it is possible to show that $DP_{RP}$ algorithm is *$\varepsilon$-optimal* in every stationary environment. To prove this, we shall prove that the $DP_{RP}$ algorithm possesses both the moderation and the monotonicity properties. As a consequence of this, the *$\varepsilon$-optimality* of this algorithm follows from the theorem presented in [5], which states that any discretized estimator algorithm that possesses both these properties is *$\varepsilon$-optimal* in every stationary environment.

*Theorem 9:* The $DP_{RP}$ algorithm possesses the monotone property. That is, if there exists an index m and a time instant $t_0 < \infty$ such that $\hat{d}_m(t) > \hat{d}_j(t)$ for all $j$, $j \neq m$ 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 1 as $t \to \infty$.

*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, based on the submartingale convergence theorem [9] we can show that there exists a random variable to which $\{p_m(t)\}_{t\geq t_0}$ converges with probability 1, which implies

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

We will show then that this leads us to the conclusion of this theorem.

We shall prove first that $\{p_m(t)\}_{t\geq t_0}$ is bounded by showing that $\sup_{t\geq 0} E[|p_m(t)|] < \infty$.

Assuming that the algorithm has not converged to the $m$th action, there exists at least one nonzero component of $\mathbf{P}(t)$, say $p_k(t)$ with $k \neq m$. Based on the premise of this theorem and the updating equations of the algorithm (14), the $p_k(t)$ component of the $\mathbf{P}(t)$ vector decreases, leading us to

$$p_k(t+1) = \max\{p_k(t) - \Delta, 0\} < p_k(t).$$

Because the $m$th component is the one with the maximum estimate, $p_m(t+1)$ increases, and we have

$$p_m(t+1) = 1 - \sum_{j\neq m} \max\{p_j(t) - \Delta, 0\} > p_m(t).$$

If $c_t$ denotes the number of $p_k(t)$ values greater than 0 at time $t$, we can quantify the increase in the value of the $p_m(t)$ component as follows:

$$p_m(t+1) = p_m(t) + c_t \cdot \Delta$$

where $\Delta = 1/rn$ is the smallest step size and $0 < c_t < r$. Therefore, we have

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

Since all the terms on the right-hand of the equation are bounded, this implies that

$$\sup_{t\geq 0} E[p_m(t+1)|Q(t), p_m(t) \neq 1] < \infty. \tag{15}$$

Also

$$E[p_m(t+1) - p_m(t)|Q(t)] = c_t\Delta \geq 0 \quad \text{for all } t \geq t_0 \tag{16}$$

implying that $\{p_m(t)\}$ is a submartingale.

By the submartingale convergence theorem [9] and based on (15) and (16), $\{p_m(t)\}_{t\geq t0}$ converges with probability 1 as $t \to \infty$, hence

$$E[p_m(t+1) - p_m(t)|Q(t)] = c_t \cdot \Delta \underset{t\to\infty}{\longrightarrow} 0 \text{ w. p. 1.}$$

This implies that $c_t \to 0$ w.p. 1, i.e., the number of components (other than $p_m(t)$) greater than 0 of the probability vector $\mathbf{P}(t)$ converges to 0 with probability 1. Thus

$$\sum_{j\neq m} \max\{p_j(t) - \Delta, 0\} \underset{t\to\infty}{\longrightarrow} 0 \text{ w.p. 1}$$

and hence $p_m(t) \to 1$ w.p. 1. This proves the theorem.

We will show next that the $\text{DP}_{\text{RI}}$ algorithm possesses the property of moderation.

*Theorem 10:* The $\text{DP}_{\text{RI}}$ algorithm possesses the moderation property.

*Proof:* To prove this theorem, we see that the magnitude by which an action probability can decrease per iteration is bounded by $1/rn$. Indeed, from the updating equations (14), we have

$$p_j(t+1) = \max\{p_j(t) - \Delta, 0\}$$

which implies that $p_j(t+1) \geq p_j(t) - 1/rn$, or equivalently $p_j(t) - p_j(t+1) \leq 1/rn$.

This means that in the worst case, when a probability decreases, it decreases by $\Delta$, which is $1/rn$.

The *$\varepsilon$-optimal* property of the algorithm follows.

*Theorem 11:* The $\text{DP}_{\text{RI}}$ is *$\varepsilon$-optimal* in all random environments.

*Proof:* The theorem follows in a straightforward manner from the moderation and monotone properties proved above, and the fundamental result stating the necessary and sufficient conditions required for an estimator algorithm to be *$\varepsilon$-optimal* [5].

## IV. Experimental Results

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

When the simulations were performed considering the same threshold $T$ and number of experiments NE as Oommen and Lanctôt did in [16], (i.e., $T = 0.99$ and $NE = 75$), the learning parameters obtained for the $\text{DP}_{\text{RI}}$ algorithm in the (0.8 0.6) environment had a variance coefficient of 0.275 666 5 in 40 tests performed. This variance coefficient was not considered satisfactory for comparing the performance of the Pursuit algorithms. Subsequent simulations were performed imposing stricter convergence requirements by increasing the threshold $T$, and proportionally, the number of experiments NE, which yielded learning parameters with smaller variance coefficients. For example, the learning parameter $(N)$ for $\text{DP}_{\text{RI}}$ algorithm in the (0.8 0.6) environment, when $T = 0.999$ and $NE = 750$, exhibits a variance coefficient of 0.070 689 3, which represents a much smaller variance. Therefore, in this paper the simulation results for $T = 0.999$ and NE equal to 500 and 750 experiments shall be presented.

The simulations were performed for different existing benchmark environments with two and ten actions. These environments have been used also to compare a variety of continuous and discretized schemes, and in particular, the $\text{DP}_{\text{RI}}$ in [16] and to compare the performance of the $\text{CP}_{\text{RI}}$ against other traditional VSSA in [23]. Furthermore, to keep the conditions identical, each estimator algorithm sampled all actions one times each in order to initialize the estimate vector. These extra iterations are also included in the results presented in the following tables. Tables I and II contain the simulation results for these four algorithms in two action environments. The probability of reward for one action was fixed at 0.8 for all simulations and the probability of reward for the second action was increased from 0.5 to 0.7.[1] In each case, the reported results correspond to the results obtained using the above-described "best" parameter.

Since the advantages that the "RI" schemes possess could be potentially overshadowed in environments when the penalty probabilities are large (i.e., when most of the responses obtained

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

TABLE I
COMPARISON OF THE PURSUIT ALGORITHMS IN TWO-ACTION BENCHMARK ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 750 EXPERIMENTS (NE = 750)

| Environ. | | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|---|
| $d_1$ | $d_2$ | N | No. of Iterat. | N | No. of Iterat. | $\lambda$ | No. of Iterat. | $\lambda$ | No. of Iterat. |
| 0.8 | 0.5 | 20 | 49.07 | 32 | 53.74 | 0.214 | 55.45 | 0.122 | 69.69 |
| 0.8 | 0.6 | 58 | 105.64 | 89 | 118.24 | 0.046 | 198.32 | 0.027 | 258.6 |
| 0.8 | 0.7 | 274 | 430.2 | 391 | 456.13 | 0.0091 | 939.63 | 0.0072 | 942.88 |

TABLE II
COMPARISON OF THE PURSUIT ALGORITHMS IN TWO-ACTION BENCHMARK ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 500 EXPERIMENTS (NE = 500)

| Environ. | | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|---|
| $d_1$ | $d_2$ | N | No. of Iterat. | N | No. of Iterat. | $\lambda$ | No. of Iterat. | $\lambda$ | No. of Iterat. |
| 0.8 | 0.5 | 17 | 44.75 | 26 | 47.908 | 0.314 | 43.128 | 0.169 | 55.504 |
| 0.8 | 0.6 | 52 | 97.008 | 74 | 102.176 | 0.054 | 171.852 | 0.036 | 199.936 |
| 0.8 | 0.7 | 217 | 357.99 | 297 | 364.09 | 0.011 | 789.296 | 0.0075 | 905.368 |

TABLE III
COMPARISON OF THE PURSUIT ALGORITHMS IN NEW TWO-ACTION ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 750 EXPERIMENTS (NE = 750)

| Environ. | | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|---|
| $d_1$ | $d_2$ | N | No. of Iterat. | N | No. of Iterat. | $\lambda$ | No. of Iterat. | $\lambda$ | No. of Iterat. |
| 0.2 | 0.5 | 12 | 51.29 | 38 | 60.16 | 0.349 | 54.25 | 0.089 | 89.25 |
| 0.2 | 0.4 | 27 | 108.12 | 100 | 129.69 | 0.109 | 172.14 | 0.0273 | 255.01 |
| 0.2 | 0.3 | 89 | 403.59 | 402 | 479.69 | 0.0294 | 797.78 | 0.005 | 1321.11 |

TABLE IV
COMPARISON OF THE PURSUIT ALGORITHMS IN TEN-ACTION BENCHMARK ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 750 EXPERIMENTS WHERE THE REWARD PROBABILITIES FOR THE ACTIONS ARE $\mathbf{E_A}$: 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2, $\mathbf{E_B}$: 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

| Environ. | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|
| | N | No. of Iterat. | N | No. of Iterat. | $\lambda$ | No. of Iterat. | $\lambda$ | No. of Iterat. |
| $E_A$ | 188 | 752.72 | 572 | 1126.81 | 0.0097 | 1230.36 | 0.003 | 2427.38 |
| $E_B$ | 1060 | 2693.79 | 1655 | 3230.37 | 0.002 | 4603.07 | 0.00126 | 5685.07 |

by the LA are penalties) we also compared the various pursuit LA in *such* environments.[2] As can be seen from the experimental results, these environments are more "challenging" to learn from because the penalties obtained far overshadow the effect of the rewards. In one sense, these environments can be seen to be "mirror reflections"of the environments mentioned above and studied in [16] and [23], because the penalty probabilities of the inferior action was kept at 0.8 and the penalty probability of the other action was varied. Again, 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 the following tables. Table III contains the simulation results for these four algorithms in two action environments. The probability of reward for one action was fixed at 0.2 for all simulations and the probability of reward for the second action was increased from 0.3 to 0.5. As in the above case, in each row, the reported results correspond to the results obtained using the above-described "best" parameter over the entire spectrum of parameters which yielded exact convergence in all $NE = 750$ experiments.

The simulations suggest that as the difference in the reward probabilities decreases (i.e., as the environment gets more difficult to learn from), the **discretized** pursuit algorithms exhibit a performance superior to the **continuous** algorithms. Also, comparing the pursuit algorithms based on the reward-inaction paradigm with the pursuit algorithms based on the reward-penalty paradigm, one can notice that, in general, the pursuit reward-inaction algorithms are up to 30% faster than the reward-penalty pursuit algorithms. For example, when $d_1 = 0.8$ and $d_2 = 0.6$, the $DP_{RI}$ converges to the correct action in an average of 105.64 iterations, and the $DP_{RP}$ algorithm converges in an average of 118.24 iterations. In the same environment, the $CP_{RI}$ algorithm takes an average of 198.32 iterations and the $CP_{RP}$ requires 258.60 iterations, indicating that it is about 30% slower than the $CP_{RI}$ algorithm. Amazingly enough, the superiority of the $DP_{RI}$ over the $CP_{RP}$ is more marked in the "more difficult" environments when the penalty probabilities are larger. Indeed, in the case of the environment with reward probabilities $d_1 = 0.2$ and $d_2 = 0.3$, the $DP_{RI}$ needed only 403.59 iterations while the $CP_{RP}$ needed 1321.11—more than *three* times the former.

To render the suite complete, similar experiments were also performed in the benchmark ten-action environments [16], [23]. The following tables, Tables IV and V, present the results obtained in these environments for the scenarios when the criterion for choosing the parameter demanded an exact convergence of 750 and 500 experiments respectively. Further, to also consider the case when the penalty probabilities were large[3] $\mathbf{E_C}$, a mirror environment of $\mathbf{E_A}$, a ten action environment, was devised. Here, the reward probability of the best action was kept at 0.3 and the reward probabilities of the other actions were scaled to have values between 0 and 0.3 so as to be of the same proportions as those in $\mathbf{E_A}$. The results are reported in Table VI.

As in the previous two-action environments, in ten-action environments, the $DP_{RI}$ algorithm proved to have the best performance, converging to the correct action almost 25% faster than the $DP_{RP}$ algorithm, and almost 50% faster than the $CP_{RP}$ algorithm. If we analyze the behavior of these automata in the first environment $E_A$ when $NE = 750$, the average number of iterations required by the $DP_{RI}$ to converge is 752.7, whereas the $DP_{RP}$ requires 1126.8, which implies that the $DP_{RI}$ algorithm is almost 30% faster than the $DP_{RI}$. In the same environment, the $CP_{RI}$ requires an average of 1230.3 iterations for convergence and the $CP_{RP}$ requires 2427.3, which shows that the $CP_{RI}$ is

[2]As mentioned in the initial acknowledgment, we are very grateful to Prof. M. A. L. Thathachar for his help in preparing this paper. In particular, we are grateful to him for suggesting these sets of experiments involving both two-action and multi-action environments. Although the conclusions are identical, these simulations were very time consuming. Since in every case, we ran the experiments for correct convergence in all NE experiments *over the entire spectrum of potential parameters*, the experiments for the ten action case took days of *dedicated* computing on a Pentium 200 processor. We have therefore reported the results only for one value of NE, which is 750.

[3]We are again grateful to Prof. Thathachar for suggesting this environment and these experiments.

TABLE V
COMPARISON OF THE PURSUIT ALGORITHMS IN TEN-ACTION BENCHMARK ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 500 EXPERIMENTS AND THE REWARD PROBABILITIES ARE THE SAME AS IN TABLE IV

| Envi ron. | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|
| | N | No. of Iterat. | N | No. of Iterat. | λ | No. of Iterat. | λ | No. of Iterat. |
| $E_A$ | 153 | 656.73 | 377 | 872.56 | 0.0128 | 970.32 | 0.0049 | 1544.17 |
| $E_B$ | 730 | 2084 | 1230 | 2511 | 0.00225 | 4126.58 | 0.00128 | 5589.54 |

TABLE VI
COMPARISON OF THE PURSUIT ALGORITHMS IN NEW TEN-ACTION BENCHMARK ENVIRONMENTS FOR WHICH EXACT CONVERGENCE WAS REQUIRED IN 750 EXPERIMENTS AND THE ACTION REWARD PROBABILITIES ARE: $\mathbf{E_C}$: 0.3.021 0.12 0.08 0.17 0.21 0.17 0.12 0.21 0.08

| Envir on. | $DP_{RI}$ | | $DP_{RP}$ | | $CP_{RI}$ | | $CP_{RP}$ | |
|---|---|---|---|---|---|---|---|---|
| | N | No. of Iterat. | N | No. of Iterat. | λ | No. of Iterat. | λ | No. of Iterat. |
| $E_C$ | 394 | 3399.20 | 2130 | 4540.46 | 0.0041 | 6441.32 | 0.00079 | 9094.28 |

50% faster than $CP_{RP}$, and the $DP_{RI}$ is almost 70% faster than the $CP_{RP}$. The results for $\mathbf{E_C}$ are identical. Notice that the $\mathbf{DP_{RI}}$ algorithm, which requires only 3399.2 iterations on the average, is almost three times faster than the $\mathbf{CR_{RP}}$ in this difficult environment, which requires 9094.28 iterations on the average.

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

**Best Algorithm :** discretized pursuit reward-inaction ($DP_{RI}$)
**Second-best Algorithm :** discretized pursuit reward-penalty ($DP_{RP}$)
**Third-best algorithm :** continuous pursuit reward-inaction ($CP_{RI}$)
**Fourth-best algorithm :** continuous pursuit reward-penalty ($CR_{RP}$)

Indeed, if we compare the algorithms quantitatively, we observe that the discretized versions are up to 30% faster than their continuous counterparts. Furthermore, if we compare the reward-inaction pursuit algorithms against the reward-penalty algorithms, we see that the reward-inaction algorithms are superior in the rate of convergence, being up to 25% faster than their reward-penalty counterparts, and in some cases even three times faster. Fig. 1 shows a graphical representation of the performance of these four algorithms in benchmark two-action environments. **The performance comparison for the ten-action environments is analogous.**

## V. CONCLUSION

Over the last decade, many new families of learning automata have emerged, with the class of estimator algorithms being among the fastest ones. Thathachar and Sastry [23], through the pursuit algorithm, introduced the concept of algorithms that pursue the current optimal action, following a *reward-penalty* learning philosophy. Later, Oommen and Lanctôt [16] extended

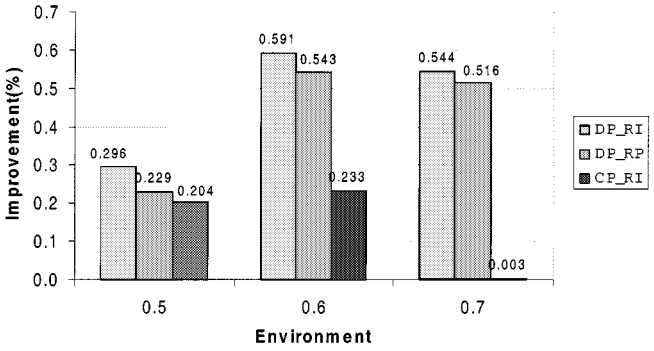


Fig. 1. Performance of the pursuit algorithms relative to the $CP_{RP}$ algorithm in two-action environments for which exact convergence was required in 750 experiments. Reward probability for the first action was considered 0.8 and the reward probability for the second action is represented on the environment axis. Performance comparison for the ten-action environments is identical.

the pursuit algorithm into the discretized world by presenting the discretized pursuit algorithm, based on a *reward-inaction* learning philosophy. In this paper we argued that a scheme that merges the pursuit concept with the most recent response of the environment, permits the algorithm to utilize the LAs *long-term* and *short-term* perspectives of the environment. Thus, we have demonstrated that the combination of the reward-penalty and reward-inaction learning paradigms in conjunction with the continuous and discrete models of computation, can lead to four versions of pursuit LAs. In this paper, we presented them all, ($DP_{RI}$, $DP_{RP}$, $CP_{RI}$, $CP_{RP}$) and also presented a quantitative comparison between them. Overall, the discrete pursuit reward-inaction algorithm surpasses the performance of all the other versions of pursuit algorithms. Also, the reward-inaction schemes are superior to their reward-penalty counterparts. Although the present comparison is solely based on rigorous experimental results involving benchmark random environments (and other "difficult" environments), it opens the door to various open problems including a formal convergence analysis of the various pursuit schemes. Such an analysis is currently being undertaken.

## ACKNOWLEDGMENT

The authors are extremely grateful to M. A. L. Thathachar of the Indian Institute of Science, Bangalore, India, for his constant input during this whole study. It was he who suggested this investigation while he was on his Sabbatical in Singapore, and gave the authors comments on the results and conclusions during the whole process.

## REFERENCES

[1] S. Baba *et al.*, "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] ——, "Two person decentralized team with incomplete information," *Appl. Math. Computat.*, 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, May 1973.
[5] J. K. Lanctôt, "Discrete estimator algorithms: A mathematical model of computer learning," M.Sc. thesis, Carleton Univ., Dept. Math. Statist., 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./Dec. 1992.
[7] M. R. Meybodi, "Learning automata and its application to priority assignment in a queuing system with unknown characteristic," Ph.D. thesis, Univ. Oklahoma, School Elect. Eng. Comput. Sci., Norman, 1983.
[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: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[10] ——, "Learning automata—A survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-4, pp. 323–334, July 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 *et al.*, "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, May/June 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.*, vol. 14, 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, pp. 931–938, July/Aug. 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] ——, "Fast object partitioning using stochastic learning automata," in *Proc. Int. Conf. Res. Dev. Inform. Retrieval*, New Orleans, LA, June 1987.
[19] B. J. Oommen and M. A. L. Thathachar, "Multiaction learning automata possessing ergodicity of the mean," *Inform. Sci.*, vol. 35, no. 3, pp. 183–198, June 1985.
[20] R. Ramesh, "Learning automata in pattern classification," M.E. thesis, Indian Inst. Sci., Dept. Elect. Eng., 1983.
[21] M. A. L. Thathachar and B. J. Oommen, "Discretized reward-inaction learning automata," *J. Cybern. Inform. Sci.*, pp. 24–29, Spring 1979.
[22] M. A. L. Thathachar and P. S. Sastry, "A class of rapidly converging algorithms for learning automata," in *Proc. IEEE Int. Conf. Cybern. Soc.*, Bombay, India, Jan. 1984, pp. 602–606.
[23] ——, "Estimator algorithms for learning automata," in *Proc. Platinum Jubilee Conf. Syst. Signal Process*. Bangalore, India, Dec. 1986, pp. 29–32.
[24] M. L. Tsetlin, "On the behavior of finite automata in random media," *Automat. Telemek.*, vol. 22, pp. 1345–1354, Oct. 1961.
[25] ——, *Automaton Theory and the Modeling of Biological Systems*. New York: Academic, 1973.
[26] V. I. Varshavskii and I. P. Vorontsova, "On the behavior of stochastic automata with variable structure," *Automat. Telemek.*, vol. 24, pp. 327–333, 1963.

**B. John Oommen** (S'79–M'83–SM'88) 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.

In 1981, he joined the School of Computer Science at Carleton University, Ottawa, ON, Canada, where he is now a Full Professor. His research interests include automata learning, adaptive data structures, statistical and syntactic pattern recognition, stochastic algorithms, partitioning algorithms, and query optimization. He is the author of more than 170 refereed journal and conference publications.

**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.

Since 1997, she has worked as a Network Software Engineer with Nortel Networks, Ottawa. Her research interests include learning machines and automata, pattern recognition, and routing algorithms.