Correspondence

Finite Time Analysis of the Pursuit Algorithm for Learning Automata

K. Rajaraman and P. S. Sastry

Abstract—**The problem of analyzing the finite time behavior of learning automata is considered. This problem involves the finite time analysis of the learning algorithm used by the learning automaton and is important in determining the rate of convergence of the automaton. In this paper, a general framework for analyzing the finite time behavior of the automaton learning algorithms is proposed. Using this framework, the finite time analysis of the Pursuit Algorithm is presented. We have considered both continuous and discretized forms of the pursuit algorithm. Based on the results of the analysis, we compare the rates of convergence of these two versions of the pursuit algorithm. At the end of the paper, we also compare our framework with that of Probably Approximately Correct (PAC) learning.**

I. Introduction

Learning Automata are adaptive decision making devices operating in unknown random environments and have been used as models of learning systems [1], [2]. Learning automata have found applications in various fields such as game theory, pattern recognition, routing in communication networks, computer vision and concept learning [2]–[6].

The learning automaton has a finite set of actions and each action has a certain probability(unknown to the automaton) of getting rewarded by the environment. The aim is to learn to choose the optimal action (i.e., the one with the highest probability of being rewarded) through repeated interaction with the environment. If the learning algorithm is chosen properly, then the iterative process (of interacting with the environment) can be made to result in the optimal action being selected with arbitrarily high probability. Several Learning Algorithms have been proposed in the literature and their asymptotic convergence properties established (see [2] for a description of many such Learning Algorithms). In addition to convergence to the optimal action, an equally important consideration is the finite time behavior of the learning algorithms. While the asymptotic analysis shows the accuracy of the automaton, the finite time analysis enables one to measure the speed of operation of the automaton. Though there are many asymptotic results available for the automata learning schemes, the results regarding their finite time behavior are very few. The main reason for this situation is the lack of a general framework for tackling the problem of finite time analysis. In this paper, we adopt one such framework and using this, present analysis of the finite time behavior of a specific learning algorithm called the pursuit algorithm [7]. The motivation behind our work is from the idea of Probably Approximately Correct (PAC) learning, as used in the Computational Learnability Theory [8]–[11]. We will discuss the similarity between the idea of PAC Learning and our analysis of finite time behavior of Learning Automata in Section VI.

Manuscript received July 2, 1992; revised July 12, 1993, and March 18, 1995.

The authors are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India (e-mails: rajram@vidyut.ee.iisc.ernet.in; sastry@vidyut.ee.iisc.ernet.in).

Publisher Item Identifier S 1083-4419(96)03929-5.

The main results of the paper are as follows. We have considered the case of learning automaton using pursuit algorithm [7], [12] under stationary environments. Both continuous and discretized versions of the pursuit algorithm have been considered. Both the algorithms are known to be ϵ-optimal [12] (see Section II for definition of ϵ-optimality). For both versions, we have derived bounds on the number of iterations and the parameter of the learning algorithm, for a given accuracy of performance of the automaton. Based on these bounds, which characterize the finite time behavior of the automaton, we compare the two versions of pursuit algorithm. Our method is useful for analyzing the finite time behavior of other estimator algorithms [7], [13], [14] as well.

The rest of the paper is organized as follows. In Section II, we describe our notation and formulate the problem of finite time behavior. In Section III, the continuous pursuit algorithm is explained and analysis of its finite time behavior is presented. Section IV contains the description of discretized pursuit algorithm and its finite time analysis. In Section V, we compare the performance of the two versions of the pursuit algorithm based on the complexity bounds obtained. Section VI concludes this paper with a discussion on the similarity between our formulation of finite time behavior and that of the PAC Learning.

II. Problem Formulation

In this section, we explain the basics of learning automata and formulate the problem of finite time behavior of their learning algorithms.

A. Learning Automata

A learning automaton is a stochastic automaton in feedback connection with a random environment [2]. The output of the automaton (called the action) is input to the environment and the output of the environment (called the reaction) is input to the automaton. The automaton is defined by (A, Q, R, T) and the environment by (A, R, D), where

- $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of all actions of the automaton. We denote by $\alpha(k)$, the action of the automaton at instant k and $\alpha(k) \in A$ for all k, $k = 0, 1, 2, \ldots$. $\alpha(k)$ denotes the output of the automaton at time instant k and this is the input to the environment(It may be noted that the automaton operates in discrete time and we use the variable k to denote the time). Thus A is the set of outputs of the automaton and is also the set of inputs to the environment.
- R is the set of reactions from the environment. We denote by $\beta(k)$ the reaction received by the automaton at instant k ($\beta(k) \in R$. $\forall k$). Thus, $\beta(k)$ denotes the actual reaction or output of the environment at time k and this is input to the automaton. Throughout this paper, we assume $\beta(k)$ to take values in a bounded interval, say, $[0, M]$. Thus, R is the set of inputs to the automaton and is also the set of outputs of the environment.
- $D = \{d_1, d_2, \ldots, d_r\}$ is the set of average reward values, where

$$d_i(k) = E[\beta(k) \mid \alpha(k) = \alpha_i]$$

If the d_i's are independent of k, the environment is said to be stationary; otherwise, it is called nonstationary. These values are unknown to the automaton.

- Q is the state of the automaton defined by

$$Q(k) = (\mathbf{p}(k), \hat{\mathbf{d}}(k))$$

where

$$\mathbf{p}(k) = [p_1(k), \ldots, p_r(k)], \quad 0 \le p_i \le 1$$
$$\sum_{i=1}^{r} p_i(k) = 1, \forall k$$

is the so-called action probability vector and

$$\hat{\mathbf{d}}(k) = [\hat{d}_1(k), \ldots, \hat{d}_r(k)]$$

is the vector of estimates of the average reward values at the k-th instant (cf. Section III).

- T is the learning algorithm that is used by the automaton to update its state. We have

$$Q(k+1) = T(Q(k), \alpha(k), \beta(k))$$

T is also called the reinforcement scheme.

The automaton functions as follows. At any instant k, the automaton chooses an action $\alpha(k)$, from the set of actions A, at random depending on its current action probability vector $\mathbf{p}(k)$ (That is, $\alpha(k)$ equals α_i with probability $p_i(k)$). This action chosen becomes input to the environment and the environment responds with a random reaction $\beta(k)$ whose expected value is d_i if $\alpha(k) = \alpha_i$. Then the automaton computes $Q(k+1)$ using the learning algorithm T. At instant $k+1$, the same cycle repeats.

The aim of the automaton is to learn to choose the optimal action, i.e., the action having the maximum average reward value. Specifically, if we denote by 'm' the index of the optimal action, then

$$d_m = \max_j \{d_j\}. \tag{2.1}$$

It is desired that the action probability corresponding to α_m (i.e., p_m) tends to one as the time k goes to infinity.

Remark 2.1:

i) Normally, in the literature [2], the state Q is defined to be equal to $\mathbf{p}$. Since in this paper our interest is in estimator algorithms we have used $Q = (\mathbf{p}, \hat{\mathbf{d}})$ as the state [13].

ii) Traditionally it is required that $\beta(k) \in [0, 1]$ so that the learning algorithm will preserve $\mathbf{p}(k)$ as a probability vector. As will be evident by the description of the algorithm later on, in pursuit algorithm, $\beta(k)$ may belong to any bounded set in the positive real line, R^+.

Remark 2.2: The learning algorithm T has a fixed internal parameter denoted by μ which decides the evolution of the state of the automaton $Q(k)$. The parameter μ determines the step size of the increase/decrease in the components of action probability vector $\mathbf{p}(k)$ at each time step through the learning algorithm. Though $\mathbf{p}(k)$ (and thus $Q(k)$) depends on the parameter μ in this way, we do not explicitly show this dependence for the sake of simplicity in notation. This is a convention followed widely in the Learning Automata literature [2].

Many criteria for evaluating the performance of learning automata have been proposed in the literature (see [2] for a full account of these criteria). A general performance index which is widely used to characterize the asymptotic behavior of learning automata is defined below.

Definition 2.1: Let m be the index of the optimal action. A learning algorithm is said to be ϵ-optimal if

$$\liminf_{k \to \infty} p_m(k) > 1 - \epsilon \quad \text{a.s}$$

for any $\epsilon > 0$, by choosing sufficiently small values of the internal parameter μ of the learning algorithm (see Remark 2.2).

Both the algorithms we analyze in this paper for finite time behavior, are known to be ϵ-optimal [12].

B. Problem Formulation

The learning problem π is the pair (L, E) where $L = (A, Q, R, T)$ is the learning automaton and $E = (A, R, D)$ is the environment as defined in the previous section. We assume E to be stationary.

Let α_m be the optimal action as defined by (2.1). We assume α_m to be unique.

The error at k-th instant, $\text{error}(k)$, is defined as the probability of not choosing α_m at that instant. That is,

$$\text{error}(k) = 1 - p_m(k).$$

We define the size (denoted by θ) of the learning problem to be the difference between the two largest average reward values.

$$\text{i.e.,} \quad \theta = d_m - \max_{j \neq m} \{d_j\}. \tag{2.2}$$

It can be noted that by this definition, problems of smaller size are more difficult.

Let Ω be the set of all learning problems π such that the size of π is at least θ_0, for some fixed $\theta_0 > 0$. Let μ denote the internal parameter of the learning algorithm. Now, the problem can be stated as follows:

Given any $\epsilon, \delta \in (0, 1)$, determine $K^* = K^*(\epsilon, \delta)$ and $\mu^* = \mu^*(\epsilon, \delta)$ such that

$$\text{Prob}\{\text{error}(k) < \epsilon\} > 1 - \delta, \ \forall k > K^* \text{ and } \forall 0 < \mu < \mu^*$$

for all problems $\pi \in \Omega$.

i.e., Determine K^* and μ^* such that

$$\text{Prob}\{p_m(k) > 1 - \epsilon\} > 1 - \delta, \ \forall k > K^*, \ \forall 0 < \mu < \mu^*$$

and

$$\forall \pi \in \Omega. \tag{2.3}$$

Remark 2.3: From Definition 2.1, it is easy to see that for any ϵ-optimal learning algorithm, the K^* and μ^* as needed by (2.3) exist. Our interest in this paper is to find explicit expressions for $K^*(\epsilon, \delta)$ and $\mu^*(\epsilon, \delta)$ so that we can get bounds on the finite time behavior.

It may be noted that the functions $K^*(\epsilon, \delta)$ and $\mu^*(\epsilon, \delta)$ will depend on the value of θ_0 which is defined as the lower bound on the size of problems in Ω.

III. Pursuit Algorithm: Continuous Case

In this section, we consider learning automata using continuous pursuit algorithm and analyze its finite time behavior.

A. The Pursuit Algorithm

Pursuit algorithm is a special type of estimator Algorithm [7]. It is simple and it converges rapidly in simulations [7], [12], [14].

We first introduce the notation used in the definition and analysis of the pursuit algorithm.

Notation:

$\mathbf{e}_i$ r-dimensional vector with i-th component unity and all others zero.

$I\{A\}$ Indicator function of event A. That is,

$$I\{A\} = \begin{cases} 1 & \text{if the event } A \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

$X_i(k)$ Total reward obtained for the i-th action till k-th instant. That is,

$$X_i(k) = \sum_{j=1}^{k-1} \beta(j) I\{\alpha(j) = \alpha_i\}.$$

$Y_i(k)$ Number of times i-th action is chosen till k-th instant. That is,

$$Y_i(k) = \sum_{j=1}^{k-1} I\{\alpha(j) = \alpha_i\}.$$

$M(k)$ The index of the average reward value estimate defined as

$$\hat{d}_{M(k)}(k) = \max_j \{\hat{d}_j(k)\}.$$

(Ties to be resolved arbitrarily)

Thus, based on the estimates of the average reward values $\hat{\mathbf{d}}(k)$ at time k, $\alpha_{M(k)}$ is the optimal action.

(Recall from Section II-A that $\alpha(k)$ is the action chosen at time k, $\beta(k)$ the reaction of the environment at time k and $\hat{d}_i(k)$ the estimate of the average reward value corresponding to action α_i at time k.)

In the following, we define the pursuit algorithm by specifying the updating of the state (as defined in Section II-A) of the automaton using the above notation.

Algorithm CPursuit:

1. **Set** $p_i(0) = 1/r$ for $1 \le i \le r$.
   Initialize $\hat{\mathbf{d}}(0)$ by picking each action a small number of times and setting $\hat{d}_i(0)$ to the average of the reactions obtained during instants when α_i was chosen, $1 \le i \le r$.
2. **Set** $k = 0$.
   Repeat
   (a. At time instant k, choose $\alpha(k)$ according to the distribution $\mathbf{p}(k)$.
   (b. Let $\alpha(k) = \alpha_i$. Then

$$X_i(k+1) = X_i(k) + \beta(k)$$
$$Y_i(k+1) = Y_i(k) + 1$$
$$X_j(k+1) = X_j(k), \quad j \neq i$$
$$Y_j(k+1) = Y_j(k), \quad j \neq i$$
$$\hat{d}_i(k+1) = \frac{X_i(k+1)}{Y_i(k+1)}, \quad 1 \le i \le r \tag{3.4}$$
$$\mathbf{p}(k+1) = \mathbf{p}(k) + \mu(\mathbf{e}_{M(k)} - \mathbf{p}(k)). \tag{3.5}$$

3. **Until** convergence.

In the above, the **Repeat**... **Until** loop is executed till one of the action probabilities is greater than, say, 0.99. Recall from Section II-A that $\mathbf{p}(\mathbf{k})$ is the action probability vector of the automaton at time k. As defined in Section II-B, μ is the internal parameter of the learning algorithm.

Since $M(k)$ is the index of the maximal reward estimate, it is easy to see from (3.5) that the action probability vector is moved in the direction of the current estimate of optimal action. In other words, the automaton *pursues* the 'current' optimal action and hence the name Pursuit Algorithm.

B. Analysis of Algorithm CPursuit

In this subsection we analyze the algorithm presented above. As discussed in Section II, our objective is to determine bounds on the number of iterations and the learning parameter to satisfy (2.3). We shall denote them by $K_c^*(\epsilon, \delta)$ and $\mu^*(\epsilon, \delta)$ respectively for the purpose of this subsection.

We first prove two lemmas that will be useful in proving our main result.

Lemma 3.1: Given any $\delta \in (0,1)$ and a positive integer N such that $6 \le N < \infty$, for each action α_i, under algorithm CPursuit,

$$\text{Prob}\left[\begin{array}{c}\alpha_i \text{ is chosen utmost} \\ N \text{ times till the } k\text{-th instant}\end{array}\right] < \delta$$
$$\forall k > K_1(N,\delta) \text{ and } \forall 0 < \mu < \mu_1(N,\delta)$$

where

$$K_1(N,\delta) = \left\lceil \frac{2N}{\ln(1/\sigma)} \ln\left[\frac{N}{\ln(1/\sigma)} \frac{1}{\sigma}\left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right]\right\rceil$$
$$\mu_1(N,\delta) = 1 - 2^{-\frac{1}{K_1(N,\delta)}}$$
$$\sigma = \frac{2r-1}{2r}$$

and

$$r = \text{number of actions of the automaton.}$$

Proof: By our notation, the random variable $Y_i(k)$ denotes the number of times action α_i is chosen up to time k. Hence, we have to show that

$$\text{Prob}[Y_i(k) \le N] < \delta. \tag{3.6}$$

Since the events $\{Y_i(k) = j\}$ and $\{Y_i(k) = s\}$ are mutually exclusive for $j \neq s$, (3.6) is equivalent to

$$\sum_{j=1}^{N} \text{Prob}[Y_i(k) = j] < \delta$$

which follows if, for all j, $1 \le j \le N$,

$$\text{Prob}[Y_i(k) = j] < \frac{\delta}{N}.$$

□

At any instant k of the algorithm, $\text{Prob}\{\alpha(k) = \alpha_i\} \le 1$. Also, under algorithm CPursuit, in any one iteration, the action probability can decrease at most by $(1-\mu)$ times. Therefore, at any iteration k of the algorithm, we have

$$\text{Prob}[\alpha(k) \neq \alpha_i] \le (1 - (1-\mu)^k p_i(0)). \tag{3.7}$$

Using these two bounds, the probability that action α_i is chosen j times during k iterations has the following upper bound (by Binomial distribution).

$$\begin{aligned}\text{Prob}\{Y_i(k) = j\} &< C_j^k (1)^j [1 - (1-\mu)^k p_i(0)]^{k-j} \\ &< k^j [1 - (1-\mu)^k p_i(0)]^{k-j}.\end{aligned}$$

Hence it is sufficient to prove

$$k^j [1 - (1-\mu)^k p_i(0)]^{k-j} < \frac{\delta}{N} \ \forall j, \quad 1 \le j \le N. \tag{3.8}$$

We have to show that (3.6) holds for all $k > K_1(N,\delta)$ and $\mu < \mu_1(N,\delta)$ (where $K_1(N,\delta)$ and $\mu_1(N,\delta)$ are as in statement of Lemma 3.1). First we shall show that (3.8) holds for all sufficiently large k, if μ can be made dependent on k as[1]

$$\mu = 1 - 2^{-\frac{1}{k}}. \tag{3.9}$$

[1] We adopted this idea from [12].

We then show that this implies (3.8) for all sufficiently small μ and a sufficiently large k. After that, we prove that Lemma 3.1 follows from this.

For the first part, under the additional condition (3.9), (3.8) simplifies to finding $K_1(N,\delta)$ such that

$$k^j \sigma^{k-j} < \frac{\delta}{N}, \ \forall j, \quad 1 \le j \le N, \quad \forall k > K_1(N,\delta) \tag{3.10}$$

where $\sigma = 1 - p_i(0)/2 = (2r-1)/2r$ since $p_i(0) = 1/r$.

Since $k/\sigma \ge 1$ (the time index k takes values $1, 2, \ldots$), (3.10) follows if

$$k^N \sigma^{k-N} < \frac{\delta}{N}, \quad \text{for } k > K_1(N,\delta). \tag{3.11}$$

Consider the function $\phi(x) = x^N \sigma^{x-N}$, defined for $x > 0$

$$\phi'(x) = \sigma^{x-N} x^{N-1} (x \ln \sigma + N).$$

Now $\phi'(x) < 0$ for

$$x > \frac{N}{\ln 1/\sigma} = N_0, \text{ say.}$$

Hence to find x_0 such that

$$\phi(x) < \delta, \ \forall x > x_0$$

it is sufficient to find $x_0 > N_0$ such that

$$\phi(x_0) < \delta.$$

We write the needed x_0 as

$$x_0 = N_0 e^a, \ a > 0. \tag{3.12}$$

Substituting $k = N_0 \cdot e^a$ in (3.11) and taking logarithms on both sides of (3.11), our problem reduces to finding 'a' such that

$$N \ln \left[\frac{N}{\ln(1/\sigma)}\right] + Na + \left(N - \frac{N \cdot e^a}{\ln(1/\sigma)}\right) \ln(1/\sigma) < \ln \frac{\delta}{N}.$$

That is

$$\ln \left[\frac{N}{\ln(1/\sigma)}\right] + a + \ln(1/\sigma) - e^a < \frac{1}{N} \ln \frac{\delta}{N}.$$

That is,

$$e^a - a > \frac{1}{N} \ln \frac{N}{\delta} + \ln \left[\frac{N}{\ln(1/\sigma)} \cdot \frac{1}{\sigma}\right].$$

That is

$$e^a - a > \ln \left[\frac{N}{\ln(1/\sigma)} \cdot \frac{1}{\sigma} \cdot \left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right]. \tag{3.13}$$

Now, consider $f(x) = e^x/2 - x$

$$f'(x) = \frac{e^x}{2} - 1 > 0 \quad \text{if } x > \ln 2$$

i.e., for $x > \ln 2$, since $f(\ln 2) > 0$

$$\frac{e^x}{2} - x > 0$$

$$\text{i.e., } \quad e^x - x > \frac{e^x}{2}. \tag{3.14}$$

By (3.14), (3.13) follows if

$$\frac{e^a}{2} > \ln \left[\frac{N}{\ln(1/\sigma)} \cdot \frac{1}{\sigma} \cdot \left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right].$$

That is

$$a > \ln \left[2 \ln \left[\frac{N}{\ln(1/\sigma)} \cdot \frac{1}{\sigma} \cdot \left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right]\right] = a^*, \text{ say.}$$

If we let a^* as the value of a in (3.12) then we will have (3.11) satisfied for all k with $K_1(N,\delta) = N_0 e^{a^*}$. However, (3.12) demands $a^* > 0$. Also, since we used (3.14) in deriving a^*, we need $a^* > \ln 2$. This is true if

$$\frac{N}{\ln(1/\sigma)} \ge e$$

$$\text{i.e., if } N \ge \frac{e}{\sigma}$$

$$\text{i.e., if } N \ge \frac{2er}{2r-1}$$

$$\text{i.e., if } N \ge 6 \quad \text{since } \frac{2r}{2r-1} \le 2.$$

Thus, (3.11) is satisfied for all $k > K_1(N,\delta)$, where

$$K_1(N,\delta) = \left\lceil \frac{N \cdot e^{a^*}}{\ln(1/\sigma)} \right\rceil = \left\lceil \frac{2N}{\ln(1/\sigma)} \ln \left[\frac{N}{\ln(1/\sigma)} \frac{1}{\sigma} \left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right]\right\rceil \quad \text{if } N \ge 6 \tag{3.15}$$

which gives the value of $K_1(N,\delta)$ to satisfy (3.11) and hence (3.8) under the additional condition given by (3.9). However, we have to show that (3.8) is satisfied for $k > K_1(N,\delta)$ and $\mu < \mu_1(N,\delta)$. In getting (3.11) we have made μ a function of k given by

$$\mu(k) = 1 - 2^{-\frac{1}{k}}. \tag{3.16}$$

To complete the proof it remains to be shown that (3.8) holds for all $k > K_1(N,\delta)$ and $\mu < \mu_1(N,\delta) = 1 - 2^{-\frac{1}{K_1(N,\delta)}}$ (cf. Statement of Lemma 3.1). For $k = K_1(N,\delta)$ and $\mu = \mu_1(N,\delta)$, (3.11) is satisfied as shown above. Hence, for $k = K_1(N,\delta)$ and $\mu < \mu_1(N,\delta)$, (3.8) is satisfied because the LHS of (3.8) decreases monotonically as μ decreases. Now if μ is fixed, for any $k_1 > k_2$, by definition of $Y_i(k)$

$$\{Y_i(k_2) > N\} \subset \{Y_i(k_1) > N\}$$

or

$$\{Y_i(k_1) \le N\} \subset \{Y_i(k_2) \le N\}$$

which implies that Prob$[Y_i(k_1) \le N] \le$ Prob$[Y_i(k_2) \le N]$.

Hence, the LHS of (3.6) is monotonically decreasing as k increases. Since (3.8) implies (3.6), we have (3.6) satisfied for all $k > K_1(N,\delta)$ and $\mu < \mu_1(N,\delta)$.

Hence Lemma 3.1 follows.

Lemma 3.2: For all i, $1 \le i \le r$, given $\epsilon, \delta \in (0,1)$

$$\text{Prob}[|\hat{d}_i(k) - d_i| > \epsilon] < \delta \quad \forall k > K_2(\epsilon,\delta) \text{ and } 0 < \mu < \mu^*(\epsilon,\delta) \tag{3.17}$$

where

$$K_2(\epsilon,\delta) = K_1\left(\max\left\{6, \left\lceil \frac{M^2}{2\epsilon^2} \ln \frac{4}{\delta} \right\rceil\right\}, \ \frac{\delta}{2}\right)$$

$$\mu^*(\epsilon,\delta) = 1 - 2^{-\frac{1}{K_2(\epsilon,\delta)}}$$

$K_1(.,.)$ is as defined in Lemma 3.1

M is the upper bound on possible values of the environmental reaction (recall $\beta \in [0, M]$).

Proof: Consider the i-th action α_i.

By definition, the estimate of d_i at k-th instant is given by (see (3.4)),

$$\hat{d}(k) = \frac{\sum_{j=1}^{k-1} I\{\alpha(j) = \alpha_i\} \cdot \beta(j)}{Y_i(k)}.$$

Let m_j^i denote the time instant at which action α_i is chosen for the j-th time, $1 \leq j \leq Y_i(k)$. Therefore

$$\hat{d}(k) = \frac{\sum_{j=1}^{Y_i(k)} \beta(m_j^i)}{Y_i(k)}. \tag{3.18}$$

It can be observed that, for fixed i, the sequence of random variables $\{\beta(m_j^i),\ j \geq 1\}$ are i.i.d. Since they are also bounded by M, by applying Hoeffding's inequality [15], we have, for any N

$$\text{Prob}\left[\left|\frac{\sum_{j=1}^{N} \beta(m_j^i)}{N} - d_i\right| > \epsilon\right] < 2\exp\left(-\frac{2N\epsilon^2}{M^2}\right). \tag{3.19}$$

We will now use (3.18), (3.19) and Lemma 3.1 to complete the rest of the proof.

Define the events

$$A = \{|\hat{d}_i(k) - d_i| > \epsilon\}$$
$$B = \{Y_i(k) > N\}.$$

By laws of probability,

$$\begin{aligned} P(A) &= P(A \mid B)P(B) + P(A/\bar{B})P(\bar{B}) \\ &< P(A \mid B) + P(\bar{B}). \end{aligned} \tag{3.20}$$

Taking $N = \lceil \frac{M^2}{2\epsilon^2} \ln(\frac{4}{\delta}) \rceil$, we get from (3.18) and (3.19)

$$\begin{aligned} P(A \mid B) &= \text{Prob}[|\hat{d}_i(k) - d_i| > \epsilon \mid Y_i(k) > N] \\ &< \frac{\delta}{2}. \end{aligned} \tag{3.21}$$

By Lemma 3.1

$$\begin{aligned} P(\bar{B}) &= \text{Prob}[Y_i(k) \leq N] \\ &< \frac{\delta}{2} \quad \text{for } k > K_0 \quad \text{and} \quad \mu < 1 - 2^{-\frac{1}{K_0}} \end{aligned} \tag{3.22}$$

where $K_0 = K_1(\max\{6, N\}, \delta/2)$.

Therefore

$$\begin{aligned} P(A) &= \text{Prob}[|\hat{d}_i(k) - d_i| > \epsilon] \\ &< \frac{\delta}{2} + \frac{\delta}{2} = \delta, \quad \text{for } k > K_2(\epsilon, \delta) \quad \text{and} \quad 0 < \mu < \mu^* \end{aligned}$$

by (3.20), (3.21) and (3.22) where

$$\begin{aligned} K_2(\epsilon, \delta) &= K_1[\max\{6, N\}, \delta/2] \\ N &= \frac{M^2}{2\epsilon^2} \ln\left(\frac{4}{\delta}\right) \\ \mu^* &= 1 - 2^{-\frac{1}{K_2(\epsilon,\delta)}}. \end{aligned}$$

Since i is arbitrary and $K_2(\epsilon, \delta)$, μ^* and $p_i(0)$ (which equals $1/r$) are independent of i, the proof of Lemma 3.1 is complete. □

We now state and prove the main result for algorithm CPursuit.

Theorem 3.1: Consider a learning automaton using algorithm CPursuit. Then, $\forall \epsilon, \delta \in (0, 1)$

$$\text{Prob}[p_m(k) > 1 - \epsilon] > 1 - \delta, \quad \text{for } k > K_c^* \text{ and } 0 < \mu < \mu^* \tag{3.23}$$

where

$$K_c^* = K' + \left\lceil \frac{\ln \frac{1}{\epsilon}}{\ln \frac{1}{1-\mu^*}} \right\rceil$$
$$K' = K_1\left(\max\left\{6, \left\lceil \frac{2M^2}{\theta^2} \ln \frac{4}{\delta} \right\rceil\right\}, \frac{\delta}{2}\right)$$
$K_1(.,.)$ is as defined in Lemma 3.1
$\mu^* = 1 - 2^{-\frac{1}{K'}}$
$\theta = d_m - \max_{j \neq m}\{d_j\}$ is the problem size defined in Section II-B.

(Recall from Section II that m is the index of the optimal action).

Proof: Define the events

$$E_1(k) = \{p_m(k) > 1 - \epsilon\}$$
$$E_2(k) = \left\{\sup_{j \geq k} \max_i |\hat{d}_i(j) - d_i| < \frac{\theta}{2}\right\}, \quad k = 1, 2, \ldots$$

Let K' be some constant (whose value will be derived later on). Then

$$\begin{aligned} \Pr[p_m(k + K') > 1 - \epsilon] &= P(E_1(k + K')) \\ &\geq P(E_1(k + K') \mid E_2(K'))P(E_2(K')). \end{aligned} \tag{3.24}$$

First, we will calculate K_0 such that

$$P(E_1(k + K') \mid E_2(K')) = 1 \quad \text{if } k > K_0. \tag{3.25}$$

Later, combining this with Lemma 3.2 we will complete the proof of the theorem.

Suppose that for some K', the event $E_2(K')$ occurs. Then, by the definition of θ and the event $E_2(K')$

$$\hat{d}_m(k) > \hat{d}_j(k), \quad \forall j \neq m \quad \text{and} \quad \text{for } k > K' \tag{3.26}$$

where m is the index of the optimal action.

Now, to find K_0 such that (3.25) holds, we need to find the number of iterations required for $p_m(k + K')$ to be greater than $1 - \epsilon$ almost surely. That is, we want

$$p_m(k + K') > 1 - \epsilon$$

or

$$\sum_{j_1 \neq m} p_{j_1}(k + K') < \epsilon. \tag{3.27}$$

By algorithm CPursuit, after the instant K' satisfying (3.26), at *each* instant, for *every* j_1, $j_1 \neq m$, $p_{j_1}(k)$ is decremented by a factor of $(1 - \mu)$ and $p_m(k)$ incremented by an amount to make the sum of the probabilities unity. Hence, we can write (3.27) as

$$\sum_{j_1 \neq m} p_{j_1}(K')(1 - \mu)^k < \epsilon. \tag{3.28}$$

Since the sum of probabilities cannot exceed unity, (3.28) is satisfied if

$$(1-\mu)^k < \epsilon$$
$$\text{i.e., if } k > \left\lceil \frac{\ln \frac{1}{\epsilon}}{\ln \frac{1}{1-\mu^*}} \right\rceil. \tag{3.29}$$

Thus, we have

$$\begin{aligned}\text{Prob}[E_1(k+K') \mid E_2(K')] &= \text{Prob}[p_m(k) > 1-\epsilon \mid E_2(K')] \\ &= 1, \quad \text{for } k > \left\lceil \frac{\ln \frac{1}{\epsilon}}{\ln \frac{1}{1-\mu^*}} \right\rceil \\ &= K_0 \quad \text{by (3.29).}\end{aligned} \tag{3.30}$$

Now, to complete the proof of the theorem, it remains to be shown that we can find a constant K' such that the event $E_2(K')$ occurs with probability greater than $(1-\delta)$(see (3.24)). But, by Lemma 3.2, this is indeed possible by choosing $K' = K_2(\theta/2, \delta)$ and $\mu^* = 1-2^{-1/K'}$.

Since ϵ and δ are arbitrary, the theorem follows.

IV. Pursuit Algorithm: Discrete Case

In Section III, we discussed the algorithm CPursuit in which the action probabilities evolved in a continuous probability space; i.e., the action probabilities were assumed to take arbitrary values in $[0, 1]$. In contrast to this, automaton learning algorithms have been proposed which discretize the probability space [16], [14]. The primary motivation for such discretized learning algorithms is to increase the speed of convergence of the optimal action probability to unity. This can be achieved, since, by the discretization process, it is possibile for the action probability to converge to unity directly, rather than approach the value unity asymptotically. For a discussion on discretized learning algorithms see [16], [12], [14].

In this section, we shall analyze the finite time behavior of discretized version of the pursuit algorithm. The so-called discretized pursuit algorithm [12], [14] is identical to its continuous counterpart except that the changes in action probability are made now in discrete steps. Therefore, the action probability $p_i(k), 1 \le i \le r$, can now assume only finitely many values. That is

$$p_i(k) \in \{0, \tau, 2\tau, \ldots, 1\}, \ \forall k$$

where τ is the so-called step size.

We define the resolution parameter n_L as

$$n_L = \frac{1}{r\tau} \tag{4.31}$$

(Recall that r is the number of actions of the automaton).

Remark 4.1: We use the resolution parameter n_L as the internal parameter of the learning algorithm. The parameter n_L which will play a role similar to the parameter μ in the continuous case, determines the step size of the increase/decrease in the components of action probability vector $\mathbf{p}(k)$ at each time step through the learning algorithm. Though $\mathbf{p}(k)$ (and thus the state of the automaton $Q(k)$) depends on the parameter n_L in this way, we do not explicitly show this dependence, as in the continuous case, for the sake of simplicity in notation.

In the following, we give the discretized pursuit algorithm using the notation introduced in Section III.

Algorithm DPursuit:

1. **Set** $p_i(0) = 1/r$ for $1 \le i \le r$
   Initialize $\hat{\mathbf{d}}(0)$ by picking each action a small number of times and setting $\hat{d}_i(0)$ to the average of the reactions obtained during instants when α_i is chosen, $1 < i < r$.
2. **Set** $k = 0$.
   Repeat
   (a. At time instant k, choose $\alpha(k)$ according to the distribution $\mathbf{p}(k)$.
   (b. Let $\alpha(k) = \alpha_i$. Then

$$\begin{aligned}X_i(k+1) &= X_i(k) + \beta(k) \\ Y_i(k+1) &= Y_i(k) + 1 \\ X_j(k+1) &= X_j(k), \quad j \ne i \\ Y_j(k+1) &= Y_j(k), \quad j \ne i \\ \hat{d}_i(k+1) &= \frac{X_i(k+1)}{Y_i(k+1)}, \quad \forall i.\end{aligned} \tag{4.32}$$

   If $p_{M(k)}(k) \ne 1$,

$$p_j(k+1) = \max\{p_j(k) - \tau, 0\}, \quad j \ne M(k)$$
$$p_{M(k)}(k) = 1 - \sum_{j \ne M(k)} p_j(k+1)$$

   Else

$$\mathbf{p}(k+1) = \mathbf{p}(k).$$

3. **Until** one of the action probabilities assumes the value unity.

A. Analysis of Algorithm DPursuit

In this subsection we analyze the algorithm DPursuit presented above. Our objective is to derive expressions for the number of iterations and the resolution parameter so that we can get bounds on the finite time behavior. We shall denote them by $K_d^*(\epsilon, \delta)$ and $N_L^*(\epsilon, \delta)$ respectively.

Similar to the case of continuous pursuit algorithm, we start the analysis by proving two lemmas.

Lemma 4.1 (cf. Lemma 3.1): Given any $\delta \in (0, 1)$ and a positive integer N such that $6 \le N < \infty$, for each action α_i, under the algorithm DPursuit

$$\text{Prob}\left[\begin{matrix}\alpha_i \text{ is chosen utmost} \\ N \text{ times till the } k\text{-th instant}\end{matrix}\right] < \delta,$$
$$\forall k > K_3(N, \delta) \text{ and } \forall n_L > N_L(N, \delta)$$

where

$$K_3(N, \delta) = \left\lceil \frac{2N}{\ln(1/\sigma)} \ln\left[\frac{N}{\ln(1/\sigma)} \frac{1}{\sigma}\left(\frac{N}{\delta}\right)^{\frac{1}{N}}\right]\right\rceil$$
$$N_L(N, \delta) = 2K_3(N, \delta)$$
$$\sigma = \frac{2r-1}{2r} \text{ and}$$
$$r = \text{ number of actions of the automaton.}$$

Proof: By our notation, the random variable $Y_i(k)$ denotes the number of times action α_i is chosen up to time k. Hence, we have to show that

$$\text{Prob}[Y_i(k) \le N] < \delta. \tag{4.33}$$

□

Now, by similarity with Lemma 3.1, the proof of Lemma 4.1 is same as that of Lemma 3.1 up to the step where the effect of underlying algorithm appears. This is the step where algorithm CPursuit was used to bound Prob$\{\alpha(k) \ne \alpha_i\}$ (see inequality (3.6) in the proof of Lemma 3.1). We modify this step now using algorithm DPursuit as follows.

At each iteration of algorithm DPursuit, the action probability can decrease by at most τ. Therefore

$$\begin{aligned}\text{Prob}[\alpha(k) \neq \alpha_i] &\leq (1 - p_i(0) + k\tau) \\ &= \left(1 - p_i(0) + \frac{k}{rn_L}\right).\end{aligned}$$

Now, using the idea of Lemma 3.1, the probability that the i-th action is selected j times during k iterations can be bounded as follows

$$\text{Prob}[Y_i(k) = j] < k^j \left[1 - p_i(0) + \frac{k}{rn_L}\right]^{k-j}.$$

So, we have to prove that

$$k^j \left[1 - p_i(0) + \frac{k}{rn_L}\right]^{k-j} < \frac{\delta}{N} \forall j, \quad 1 \leq j \leq N. \tag{4.34}$$

To prove that (4.33) holds for all $k > K_3(N, \delta)$ and $n_L > N_L(N, \delta)$ (where $K_3(N, \delta)$ and $N_L(N, \delta)$ are as in statement of Lemma 4.1), we shall first show that (4.34) holds for all sufficiently large k, if n_L can be made dependent on k as

$$n_L = 2k. \tag{4.35}$$

After that, we show that this implies (4.34) for a sufficiently large k and for all sufficiently large n_L. Then we prove that Lemma 4.1 follows from this.

For the first part, under the additional condition (4.35), (4.34) simplifies to finding $K_3(N, \delta)$ such that

$$k^j \sigma^{k-j} < \frac{\delta}{N}, \ \forall j, 1 \leq j \leq N, \quad \forall k > K_3(N, \delta) \tag{4.36}$$

where

$$\sigma = 1 - p_i(0) + \frac{k}{rn_L}.$$

Comparing (4.36) with (3.10) in the proof of Lemma 3.1, we get,

$$\begin{aligned}K_3(N, \delta) &= K_1(N, \delta) \\ &= \left\lceil \frac{2N}{\ln(1/\sigma)} \ln\left[\frac{N}{\ln(1/\sigma)} \frac{1}{\sigma}\left(\frac{N}{\delta}\right)\right]\right\rceil.\end{aligned} \tag{4.37}$$

Now we have to show that (4.34) is satisfied for $k > K_3(N, \delta)$ and $n_L > N_L(N, \delta)$. In getting (4.37) from (4.34) we have made n_L a function of k given by (4.35).

To complete the proof it remains to be shown that (4.34) holds for all $k > K_3(N, \delta)$ and $n_L > N_L(N, \delta) = 2K_3(N, \delta)$ (cf. Statement of Lemma 4.1). For $k = K_3(N, \delta)$ and $n_L = N_L(N, \delta)$, (4.36) is satisfied as shown above. Hence, for $k = K_3(N, \delta)$ and $n_L > N_L(N, \delta)$, (4.34) is satisfied because the LHS of (3.8) decreases monotonically as n_L increases. Now if n_L is fixed, for any $k_1 > k_2$, by definition of $Y_i(k)$

$$\{Y_i(k_2) > N\} \subset \{Y_i(k_1) > N\}$$

or

$$\{Y_i(k_1) \leq N\} \subset \{Y_i(k_2) \leq N\}$$

which implies that $\text{Prob}[Y_i(k_1) \leq N] \leq \text{Prob}[Y_i(k_2) \leq N]$.

Hence, the LHS of (4.33) is monotonically decreasing as k increases. Now, by observing that (4.34) implies (4.33), Lemma 3.2 follows.

Lemma 4.2: For all i, $1 \leq i \leq r$, given $\epsilon, \delta \in (0, 1)$

$$\begin{aligned}&\text{Prob}[|\hat{d}_i(k) - d_i| > \epsilon] < \delta, \\ &\quad \forall k > K_4(\epsilon, \delta) \quad \text{and} \quad n_L > N_L'(\epsilon, \delta)\end{aligned} \tag{4.38}$$

where

$$K_4(\epsilon, \delta) = K_3\left(\max\left\{6, \left\lceil \frac{M^2}{2\epsilon^2} \ln \frac{4}{\delta} \right\rceil\right\}, \frac{\delta}{2}\right)$$

$$N_L'(\epsilon, \delta) = 2K_4(\epsilon, \delta)$$

$K_3(\ldots)$ is as defined in Lemma 4.1

Proof: The proof is identical to that of Lemma 3.2 except that Lemma 4.1 is used instead of Lemma 3.1 to bound $\text{Prob}\{Y_i(k) \leq N\}$ (see (3.22) in the proof of Lemma 3.1). Hence, we omit the proof.

We now state and prove the main result for algorithm DPursuit.

Theorem 4.1: Consider a learning automaton using algorithm DPursuit. Then, $\forall \epsilon, \delta \in (0, 1)$

$$\begin{aligned}&\text{Prob}[p_m(k) > 1 - \epsilon] > 1 - \delta, \\ &\quad \text{for} \quad k > K_d^* \quad \text{and} \quad n_L > N_L^*.\end{aligned} \tag{4.39}$$

$$K_d^* = K' + \lceil rN_L^*(1 - \epsilon) \rceil$$

$$K' = K_3\left(\max\left\{6, \left\lceil \frac{2M^2}{\theta^2} \ln \frac{4}{\delta} \right\rceil\right\}, \frac{\delta}{2}\right)$$

$K_3(\ldots)$ is as defined in Lemma 4.1

$N_L^* = 2K'$

$\theta = d_m - \max_{j \neq m}\{d_j\}$, is the problem size defined in Section II-B.

Proof: By similarity with Theorem 3.1, we need to find only the constant K_0 such that

$$p_m(k + K') > 1 - \epsilon \quad \text{for } k > K_0 \tag{4.40}$$

is true, given the event (denoted by $E_2(K')$ in Theorem 3.1)

$$\left\{\sup_{k \geq K'} \max_i |\hat{d}_i(k) - d_i| < \frac{\theta}{2}\right\}$$

occurs. The rest of the proof follows by using Lemma 4.2 instead of Lemma 3.2 in Theorem 3.1.

Let the event $E_2(K')$ occur. To find K_0 to satisfy (4.40), we need to find k such that

$$p_m(k + K') > 1 - \epsilon$$

or

$$\sum_{j_1 \neq m} p_{j_1}(k + K') < \epsilon. \tag{4.41}$$

After the instant K', by algorithm DPursuit, all action probabilities $p_{j_1}(k), j_1 \neq m$, are decremented by a factor of τ or set to zero if the decremented value becomes negative. Hence, we can write (4.41) as

$$\sum_{j_1 \neq m} \max\left\{p_{j_1}(K') - \frac{k}{rn_L}, 0\right\} < \epsilon \tag{4.42}$$

Since only one probability may actually be decremented, (4.42) is satisfied if

$$\begin{aligned}&\sum_{j_1 \neq m} [p_{j_1}(K')] - \frac{k}{rn_L} < \epsilon, \\ &\text{i.e., if } 1 - \frac{k}{rn_L} < \epsilon, \\ &\text{i.e., if } k > \lceil rn_L(1 - \epsilon) \rceil.\end{aligned}$$

Therefore,

$$p_m(k) > 1 - \epsilon \quad \text{w.p. 1 for } k > K' + \lceil rn_L(1-\epsilon) \rceil.$$

Now, to complete the proof of the theorem, it remains to be shown that we can find a constant K' such that the event $E_2(K')$ occurs with probability greater than $(1 - \delta)$. But, by Lemma 4.2, this is indeed possible by choosing $K' = K_4(\theta/2, \delta)$ and $N_L^* = 2K'$.

Hence, the proof of the theorem is complete.

V. Discussion

In Sections III and IV, we obtained explicit bounds on the finite time behavior of a learning automaton operating in a stationary environment using pursuit algorithm. In this section, the continuous and discretized versions of the pursuit algorithm are compared based on these bounds.

The finite time behavior is obtained in Sections III and IV in terms of the bounds K_c^*, μ^* (for algorithm CPursuit), K_d^* and N_L^* (for algorithm DPursuit) such that

$$\text{Prob}[p_m(k) > 1 - \epsilon] > 1 - \delta, \quad \forall k > K_c^* \text{ and } \forall \mu < \mu^*, \text{ under algorithm CPursuit.} \tag{5.43}$$

$$\text{Prob}[p_m(k) > 1 - \epsilon] > 1 - \delta, \quad \forall k > K_d^* \text{ and } \forall n_L > N_L^*, \text{ under algorithm DPursuit.} \tag{5.44}$$

The bounds depend on ϵ, δ, and the parameter of the learning problem given by M (the upper bound on the value of environmental reaction), r (the number of available actions) and θ (the difference between the two largest reward probabilities). θ is a measure of the complexity of the learning problem that is being solved by the automaton and is termed the size of the problem.

For the purpose of comparing the two pursuit algorithms, the values of above-mentioned bounds are computed numerically by considering the following three cases

1) Fixing the problem size (θ), the allowed error in selecting the optimal action (ϵ) and varying the error probability (δ).
2) Fixing the allowed error (ϵ), the error probability (δ) and varying the problem size (θ).
3) Fixing the problem size (θ), the error probability (δ) and varying the allowed error (ϵ).

The numerical results are tabulated in Tables I–III.

It is noted from the tables that the number of iterations needed for convergence are within an order of magnitude of the actual number of iterations observed in practice through simulations [7], [13], [12]. It may be mentioned that the reason why the theoretically computed value for the number of iterations is higher, is that the theoretically allowed value of $\mu^*(N_L^*)$ is lower (higher) than what is used in practice. However, the bounds give a reasonable idea of the rate of convergence of the learning algorithm.

Numerical Results: Let $\beta \in [0, 1]$ and the number of actions be 2. Therefore, $M = 1$ and $\sigma = 0.75$.

Let

$$N = \left\lceil \frac{2M^2}{\theta^2} \ln \frac{4}{\delta} \right\rceil$$

$$K = K_1\left(N, \frac{\delta}{2}\right) \left(= K_3\left(N, \frac{\delta}{2}\right)\right)$$

(see statements of Lemmas 3.1 and 4.1])

$$K_c = \left\lceil \frac{\ln \frac{1}{\epsilon}}{\ln \frac{1}{1-\mu}} \right\rceil$$

$$K_d = \lceil rN_L^*(1-\epsilon) \rceil$$

Case (i) $\theta = 0.5$; $\epsilon = 0.1$.

TABLE I

δ	N	K	μ^* $\times 10^{-4}$	K_c	$K_c^* = K + K_c$	N_L^*	K_d	$K_d^* = K + K_d$
0.100	29	1053	6.58	3498	4551	2106	3791	4843
0.080	31	1129	6.13	3755	4884	2258	4064	5193
0.050	35	1290	5.37	4286	5576	2580	4644	5934
0.001	66	2708	2.56	8995	11703	5416	9748	12456

Case (ii) $\epsilon = 0.1$; $\delta = 0.1$.

TABLE II

θ	N	K	μ^* $\times 10^{-4}$	K_c	$K_c^* = K + K_c$	N_L^*	K_d	$K_d^* = K + K_d$
0.500	29	1053	6.58	3498	4551	2106	3791	4843
0.400	46	1762	3.93	5854	7616	3524	6343	8105
0.300	82	3437	2.01	11420	14858	6874	12376	15814
0.200	184	8689	0.80	28866	37555	17378	31280	39969

Case (iii) $\theta = 0.5$; $s = 0.1$

$$N = 29$$

$$K = K_1\left(N, \frac{\delta}{2}\right) = 1053$$

$$\mu^* = 6.58 \times 10^{-4}$$

$$N_L^* = 2106.$$

TABLE III

ϵ	K_c	$K_c^* = K + K_c$	K_d	$K_d^* = K + K_d$
0.1	3458	4551	3791	4854
0.01	6996	8049	4170	5223
0.001	10494	11547	4207	5260
0.0001	13992	15045	4211	5264
0.00001	17491	18544	4212	5265

In Cases (i) and (ii), both the algorithms performed similarly when the error probability/problem size is varied.

In Case (iii), the CPursuit showed a logarithmic increase in the number of iterations required for convergence (i.e., K_c^*) as the allowed error is decreased. But, for the same decrease in the allowed error, the number of iterations needed in the case of DPursuit (K_d^*) showed a saturating behavior. This is not surprising since, in DPursuit, it is possible for the action probability to converge to unity in finite number of iterations as the probablity space has been discretized. However, in CPursuit, the action probability can converge to unity only asymptotically as the number of iterations tends to ∞ and so K_c^* does not saturate.

VI. Conclusion

We considered the problem of estimating the finite time behavior of learning automata. This problem is important in determining the rate

of convergence of the automaton. A general framework for analyzing the finite time behavior of the automaton learning algorithms was proposed. Using this framework, the finite time analysis of a specific algorithm, called the pursuit algorithm, was presented. For our analysis, we considered both continuous and discretized forms of the pursuit algorithm. Based on the results of the analysis, the rates of convergence of these two versions of the pursuit algorithm were compared.

The motivation behind the framework in which the analysis is carried out in this paper, is the concept of Probably Approximately Correct (PAC) Learning. Here we compare our framework with that of PAC Learning.

The concept of PAC learning can be described informally as follows. The *learner* wishes to learn an unknown concept. The *teacher* provides the learner with random pre-classified examples drawn independently from some (unknown) probability distribution. After observing finite number of examples, the learner outputs a hypothesis which is his current estimate of the concept. The error of the hypothesis is taken as the probability that it incorrectly classifies a random example. In this framework a concept class C is said to be PAC-learnable if the following two conditions are satisfied uniformly for all concepts in C:

- With high probability, the learner outputs a hypothesis that has arbitrarily small error after finite number of examples.
- the number of examples required is bounded independent of the distribution with which the examples are drawn.

The minimum number of examples required for the concept class C to be PAC-learnable to a given accuracy, is called the sample complexity of C. An important aspect of this formulation is the notion of a measure of difficulty for the concept class C (the so-called VC-dimension [9]) that determines the sample complexity of C.

For a precise formulation of this framework, the reader is referred to [9].

We can establish the following analogy between the PAC-learning framework and the automata learning framework. In the latter, the learner is represented by the automaton and the teacher by the environment. The learner here wishes to learn the identity of the optimal action (the unknown *concept*) under a fixed set of reward probabilities. At every iteration of the learning process, the teacher provides the learner with a reward for the action chosen (the *example*). Using this, the learner outputs a *hypothesis* which is the action chosen by the automaton.

We defined the error (see Section II) of the hypothesis at the k-th iteration as the probability of not choosing the optimal action at that iteration. This is also seen to be analogous to that in PAC setup since this gives the probability of committing a mistake at the next instant.

By denoting Ω to be a class of automaton learning problems, we can now say Ω is learnable if

- the probability of the automaton not choosing the optimal action can be made arbitrarily small with arbitrarily high probability after finite number of iterations using a suitable internal parameter of the automaton for each problem $\pi, \pi \in \Omega$.
- the number of iterations and the internal parameter required can be bounded uniformly for all problems in Ω.

The learning complexity of Ω is measured by the bounds on the number of iterations and the internal parameter that satisfy the above criteria. We have used this learning complexity measure to describe quantitatively the speed of convergence of the learning automaton.

Remark 6.1: Despite the above analogy, there is a notable difference between the two frameworks with respect to the learning methodology used. Learning in PAC-learning framework is supervisory whereas in automata models it is by reinforcement. Further, as the reinforcement signal is probabilistic, the automaton learns through noisy examples.

In analogy with PAC-learning we have characterized the difficulty of a class of Automaton Learning Problems Ω by the size parameter θ. Suppose Ω to be the collection of learning problems involving all possible environments (i.e., there is no restriction on the set of reward probabilities). We can see that Ω is not learnable because we can choose a set of sufficiently close reward probabilities such that any finite bound on the learning complexity of Ω is exceeded. Therefore, to get a meaningful theoretical bound on the learning complexity, we must restrict the class Ω. We did this by associating a *size* parameter with Ω. Analyzing complexity by such a restriction of learning problems amounts to finding rate of convergence of the automaton for problems of same difficulty.

As an extension of the work outlined in this paper, we are currently working on the finite time analysis of the L_{R-I} algorithm [2].

REFERENCES

[1] M. L. Tsetlin, *Automata Theory and Modeling of Biological Systems*. New York: Academic, 1973.

[2] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[3] K. S. Narendra, E. Wright, and L. G. Mason, "Applications of learning automata to telephone traffic routing," *IEEE Trans. Syst., Man, Cybern.*, vol. 7, pp. 785–792, 1977.

[4] M. A. L. Thathachar and P. S. Sastry, "Learning optimal discriminant functions through a cooperative game of automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 17, no. 1, pp. 73–85, 1987.

[5] ——, "Relaxation labeling with learning automata," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, pp. 256–268, Mar. 1986.

[6] P. S. Sastry and K. Rajaraman, and S. R. Ranjan, "Learning optimal conjunctive concepts through a team of stochastic automata," to appear in *IEEE Trans. Syst., Man, Cybern.*.

[7] P. S. Sastry, *Systems of Learning Automata—Estimator Algorithms and Applications*, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore, June 1985.

[8] L. G. Valiant, "A theory of the learnable," *Comm. ACM*, vol. 27, pp. 1134–1142, 1984.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and H. Warmuth, "Learnability and Vapnik-Chervonenkis dimension," *J. ACM*, vol. 36, no. 3, pp. 939–965, 1989.

[10] S. R. Kulkarni, *Problems of Computational and Information Complexity in Machine Vision and Learning*, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, 1992.

[11] D. Haussler, "Quantifying inductive bias: AI Learning algorithms and Valiant's learning framework," *Artif. Intell.*, vol. 36, pp. 177–221, 1988.

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

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

[14] J. K. Lanctot and B. J. Oommen, "Discretized estimator learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1473–1483, 1992.

[15] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Stat. Assoc.*, vol. 58, pp. 13–30, 1963.

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