DISCRETIZED ESTIMATOR LEARNING AUTOMATA[*]

J. Kevin Lanctôt[#] and B. John Oommen[+]

ABSTRACT

BOUNDED MEMORY PROBABILISTIC MOVE-TO-FRONT OPERATIONS

The concept of performing probabilistic move operations on an accessed element is not entirely new. Kan and Ross [9] suggested a probabilistic transposition scheme and showed that no advantage was obtained by rendering the scheme probabilistic. Their scheme, however, required that the probability of performing the operation, be time invariant. As opposed to this, we shall define move operations which are essentially probabilistic, but the probabilities associated with the move operations are dynamically varied.

Let f(n) be the probability (at time 'n') of any element being moved to the front of the list on being accessed. Observe that this implies that an element, on being accessed, stays where it is with probability (1-f(n)). For an initial condition we define,

$$f(0) = a \tag{2.1}$$

The probability f(n) is updated every time any record is accessed. The updating scheme is given by (2.2) below, for $0 < a < 1$.

$$f(n+1) = a\ f(n) \quad \text{every time a record is accessed.} \tag{2.2}$$

The quantity 'a' is defined as the updating constant.

Let $p_1(n)$ be the expected probability of record R_i succeeding R_j at the n^{th} time instant. Clearly $p_2(n) = 1-p_1(n)$, for all n. We shall derive the transient and asymptotic properties of $p_1(n)$. To do this we need the following lemma.

---

[*] Partially supported by the Natural Sciences and Engineering Research Council of Canada.
[#] Address of the first author : Mitel Corporation, 350 Legget Drive, Kanata, ONT : K2K 1X3, Canada. Address of the second author : School of Computer Science, Carleton University, Ottawa : ONT : K1S 5B6, Canada.
[+] Senior Member, IEEE.

LEMMA I

Let A be any nxn matrix with distinct eigenvalues. Let K be the matrix which diagonalizes A. Let

$$B(n) = I + a^n A$$

Then, B(n) is diagonalizable by the same matrix K, for all n.

Proof

Since A has distinct eigenvalues, and K diagonalies A,

$$K^{-1} A K = \mathrm{Diag}(\theta_1,...,\theta_N)$$

where $\mathrm{Diag}(\theta_1,...,\theta_N)$ is the diagonal matrix with the eigenvalues of A on its diagonal. Let $B(n) = I+a^n A$. Then,

$$\begin{aligned} K^{-1} B(n) K &= K^{-1} (I+a^n A) K \\ &= I + a^n K^{-1} A K \\ &= I + a^n \cdot \mathrm{Diag}(\theta_1,...,\theta_N) \end{aligned}$$

and the lemma is proved.
Using the above lemma and the theory of Markov's chains we prove the following theorems.

THEORM I

Let $p_1(n)$ be the expected probability of R_i succeeding R_j at the nth time instant. Then, $p_1(n)$ and $p_2(n)$ obey the following time varying Mark

[insert]

$$\begin{bmatrix} p_1(n+1) \\ p_2(n+1) \end{bmatrix} = [B(n)] \begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix}$$

where $B(n) = \begin{bmatrix} 1-a^n s_i & a^n s_i \\ a^n s_j & 1-a^n s_j \end{bmatrix}^T$

Proof

R_i succeeds R_j at time instant 'n+1' if and only if

(a) R_i succeeded R_j at time instant 'n' and no list operation was performed, or,

(b) R_j was accessed and it was moved to the front of the list. Observe that R_i cannot succeed R_j if R_i was accessed and moved to the front.

let $p_1(n)$ + Prob[R_i succeeds R_j at time 'n']. Clearly, $E[p_1(n)] = p_1(n)$. The above leads to the following recursive definition of $p_1(n)$.

$$p_1(n+1) = \begin{cases} 0 & \text{if } R_i \text{ accessed and MTF performed} \\ 1 & \text{if } R_j \text{ accessed and MTF performed} \\ p_1(n) & \text{otherwise.} \end{cases}$$

Observe that the probabilities of the events defined above are readily available in terms of the unknown access probabilities. Further, a MTF operation is performed at 'n' with a probability a^n. Thus,

$$p_1(n+1) = \begin{cases} 0 & \text{w.prob. } s_i.a^n \\ 1 & \text{w.prob } s_j\ a^n \\ p_1(n) & \text{w.prob } 1-(s_i+s_j)\ a^n \end{cases} \qquad (2.3)$$

Taking conditional expectations, we have,

$$E[p_1(n+1)|p_1(n)] = a^n\ s_j + p_1(n) - p_1(n).a^n\ (s_i + s_j)$$

Taking expectations again and observing that $E[p_1(n)]=p_1(n)$, we get,

$$p_1(n+1) = [1-a^n\ (s_i + s_j)]\ p_1(n) + s_j\ .\ a^n$$

Since $p_1(n) + p_2(n) = 1$, we expand the constant term as

$$s_j a^n = s_j a^n [p_1(n) + p_2(n)]$$

Thus,

$$p_1(n+1) = [1-a^n s_i] p_1(n) + [a^n s_j] p_2(n)$$

This leads to the following matrix equation

$$\begin{bmatrix} p_1(n+1) \\ p_2(n+1) \end{bmatrix} = \begin{bmatrix} 1-a^n s_i & a^n s_j \\ a^n s_i & 1-a^n s_j \end{bmatrix} \begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix}$$

and the theorem is proved.

THEOREM II

The constant matrix K, where

$$K = \begin{bmatrix} 1 & 1 \\ s_i/s_j & -1 \end{bmatrix}$$

diagonalizes B(n), for all values of n.

Proof

Observe that B(n) is of the form,

$$B(n) = I + a^n A$$

where

$$A = \begin{bmatrix} -s_i & s_j \\ s_i & -s_j \end{bmatrix}$$

Since B(n) is a stochastic matrix, we know that one of its eigenvalues is unity. Further, since the sum of the eigenvalues is equal to the trace of the matrix, the second eigenvalue is $1\text{-}a^n$ ((s_i+s_j). Using Lemma I, we know that the matrix which diagonalizes B(0) also diagonalizes B(n) for all n. In this case, it is easy to see that the eigenvectors for B(0) are

(i) $[1 \;\; s_i/s_j]^T$ for the eigenvalue unity, and

(ii) $[\,1 \;\; -1]^T$ for the eigenvalue $1\text{-}\,(s_i+s_j)$

Thus, the constant matrix K, where,

$$K = \begin{bmatrix} 1 & 1 \\ s_i/s_j & -1 \end{bmatrix}$$

diagonalizes B(n) for all n. This proves the theorem.

Remark: By performing elementary operations, it is easy to see that K^{-1} has the form:

$$K^{-1} = \frac{1}{s_i+s_j} \begin{bmatrix} s_j & s_j \\ s_i & -s_j \end{bmatrix} \tag{2.4}$$

One can trivially verify that,

$$K^{-1}\; B(n)\, K = \text{Diag}(1,1\text{-}a^n\,(s_i+s_j)),$$

where,

$$\text{Diag}(1,1\text{-}a^n\,(s_i+s_j)) = \begin{bmatrix} 1 & 0 \\ 0 & 1\text{-}a^n\,(s_i+s_j) \end{bmatrix}$$

similarly, K. $\text{Diag}(1,1\text{-}a^n\,(s_i+s_j))$. K^{-1} is exactly B(n).

THEOREM III

The value of $p_1(n)$ for an updating constant 'a' obtained by solving the Markov equation given by Theorem I, has the form:

$$p_1(n) = \frac{s_j}{s_i+s_j} Q_{a,n} + \frac{s_j}{s_i+s_j} (1-Q_{a,n})$$

where

$$Q_{a,n} = 0.5 \left[\prod_{k=0}^{n-1} (1-a(s_i+s_j)) \right]$$

Proof

From the results of Theorem I, we can see that

$$\begin{bmatrix} p_1(n+1) \\ p_2(n+1) \end{bmatrix} = B(n) \begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix}$$

Thus, the solution of the matrix difference equation yields,

$$\begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix} = B(n-1)\, B(n-2) \ldots B(0) \begin{bmatrix} p_1(0) \\ p_2(0) \end{bmatrix} = \prod_{k=0}^{n-1} B(k) \begin{bmatrix} p_1(0) \\ p_2(0) \end{bmatrix}$$

Rewriting each B(k) in terms of the diagonal matrix $\mathrm{Diag}(1,1-a^k (s_i+s_j))$,

$$\begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix} = \left(\prod_{k=0}^{n-1} K. [\mathrm{Diag}(1,1-a^k (s_i+s_j))] . K^{-1} \right). \begin{bmatrix} p_1(0) \\ p_2(0) \end{bmatrix}$$

Since the product of each consecutive pair K.K-1 yields that identity matrix, we write,

$$\begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix} = K. \left[\prod_{k=0}^{n-1} \mathrm{Diag}(1,1-a^k (s_i+s_j)) \right] . K^{-1} . \begin{bmatrix} p_1(0) \\ p_2(0) \end{bmatrix}$$

$$= K. \begin{bmatrix} 1 & 0 \\ 0 & \prod_{k=0}^{n-1} (1-a^k (s_i+s_j)) \end{bmatrix} .K^{-1}. \begin{bmatrix} p_1(0) \\ p_2(0) \end{bmatrix} \qquad (2.5)$$

Let $Q_{a,n} = 0.5 \prod_{k=0}^{n-1} (1-a^k (s_i+s_j))$. Since, with no loss of generality, $p_1(0) = p_2(0) = 0.5$, we expand (2.5) above to yield,

$$\begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix} = \frac{1}{s_i+s_j} \begin{bmatrix} 1 & 1 \\ \frac{s_i}{s_j} & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2.Q_{a,n} \end{bmatrix} \begin{bmatrix} s_j & s_j \\ s_i & -s_j \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

After considerable simplification this results in

$$\begin{bmatrix} p_1(n) \\ p_2(n) \end{bmatrix} = \frac{1}{s_i+s_j} \begin{bmatrix} s_j (1-Q_{a,n}) + s_i & Q_{a,n} \\ s_i (1-Q_{a,n}) + s_j & Q_{a,n} \end{bmatrix} \qquad (2.6)$$

and the theorem is proved.

Remark: Observe that $p_1(n) = p_2(n) = 0.5$ if $s_i = s_j$. This is intuitively satisfying.

To prove the asymptotic value of $p_1(n)$ we need the following lemma.

Lemma II

The infinite product:

$$\prod_{k=1} (1+b_k) \quad (\text{where } b_k \quad -1 \text{ for all } k)$$

tends to a <u>non-zero finite</u> limit if and only if the infinite sum,

$$\sum_{k=1} b_k$$

is convergent.

Proof

The lemma is proved in Titchmarsh [17, pp.13-15].

THEOREM IV

The stochastic bounded memory Move-to-Front Algorithm is asymptotically always less accurate than the deterministic Move-to-Front Algorithm.

Proof

Consider the term for $p_1(n)$ as

$$p_1(n) = \frac{s_i}{s_i+s_j}(1-Q_{a,n}) + \frac{s_j}{s_i+s_j} \; Q_{a,n}$$

where $Q_{a,n}$ is defined in Theorem III.

Using Lemma II, it is clear that $Q_{a,n} > 0$ as $0 < a < 1$, but tends to zero as a tends to unity. Differentiating with respect to a, we obtain,

$$\frac{\delta i\, p_j(n)}{\delta a} = (s_j-s_i) \,.\, Q_{a,n} \,\cdot\, {}^{\;n-1}_{\;k=1} \frac{k.a^{k-1}}{1-a^k(s_i+s_j)}$$

Since $0 < a < 1$, and $0 < s_i+s_j < 1$, we have,

$$0 < 1-a^k(s_i+s_j) < 1 \qquad \text{for all } k \quad 1.$$

Assume that with no loss of generality that $s_i < s_j$. This tells us that,

$$\frac{\delta i\, p_j(n)}{\delta a} > 0 \tag{2.7}$$

In other words, $p_1(n)$ has no stationary point with respect to a in the interval $0 < 1 < 1$. Further, due to (2.7), the largest value of $p_1(n)$ occurs when a=1. Since (2.7) is true for finite and infinite values of n, the value of $p_1(\ )$ is maximized at the largest acceptable value of a and the theorem is proved.

Corollary IV.1

The stochastic bounded memory Move-to-Front Algorithm is expedient independent of the access distribution of the records.

Proof

Due to the multiplying factor of 0.5, and the previous theorem, it is easy to see that for all a, $0 < Q_{a,n} < 0.5$. The result is now obvious since $p_1(n)$ is merely a convex combination of $\frac{s_i}{s_i+s_j}$ and $\frac{s_j}{s_i+s_j}$ weighted by $1-Q_{a,n}$ and $Q_{a,n}$ respectively.

Remark: Throughout this discussion it was assumed that the single memory location that stores f(n) can contain an arbitrarily small positive real number. In practice, however, all that we need to store is an index, n, of the time that has lapsed since the file reorganization scheme was initiated. From this index, f(n) can be computed trivially, since,

$f(n) = a^n$

It is also appropriate to observe that the maximum number that this index should attain is governed by the uniform random number generator accessible to the system. If the smallest positive number yielded by the random number generator is x_{min}, then the memory location which stores n need not store numbers larger than n_{max}, where,

$$n_{max} = \cup\log_a (x_{min})'$$

We now proceed to study the linear memory stochastic Move-to-Rear scheme.

Learning automata are stochastic automata interacting with an unknown random environment. The fundamental problem is that of learning, through feedback, the action which has the highest probability of being rewarded by the environment. A class of algorithms known as Estimator Algorithms are presently among the fastest known. They are characterized by the use of running estimates of the probabilities of each possible action being rewarded. This paper investigates the improvements gained by rendering the various estimator algorithms discrete. This is done by restricting the probability of selecting an action to a finite, and hence, discrete subset of [0,1]. This modification is proven to be ε-optimal in all stationary environments. In the body of the paper we shall first construct various Discretized Estimator Algorithms (DEAs). Subsequently, members of the family of DEAs will be shown to be ε-optimal by deriving two sufficient conditions required for the ε-optimality. Algorithms satisfying these conditions are said to possess the properties of *monotonicity* and *moderation*. We conjecture the necessity of these conditions for ε-optimality too. Experimental results indicate that the discrete modifications improve the performance of these algorithms. We believe that these automata constitute the fastest converging and most accurate learning automata reported to date.

I. INTRODUCTION

1.1 Artificial Intelligence and Learning Automata

The concept of a learning automaton (LA) was developed by Tsetlin. His intention was to model biological learning [32,33]. The learning automaton is an automaton interacting in a random environment. The LA selects an action from a finite set of possible actions. Feedback from the environment tells the LA if the chosen action was rewarded or penalized. The LA uses this information to decide which action to take next, and the cycle repeat itself. Learning automata and their applications have been reviewed by Lakshmivarahan [3], and by Narendra and Thathachar [12, 13].

Variable structure stochastic automata (VSSA) were developed by Varshavskii and Vorontsova. For these automata, the learning process is generalized so that the state transition probabilities and the action selecting probabilities evolve with time [12]. The automaton is simplified in the sense that each state now corresponds uniquely to a particular action. Hence while in state q_i the automaton always picks action α_i, and consequently, the set of actions and states are identical. Thus, what remains is the set of actions (or output from the automaton), the set of inputs (one of which serves as the input to the automaton at any time instant) and a learning algorithm T. The learning algorithm operates on a probability vector $\mathbf{P}(t) = \left(p_1(t), p_2(t), \ldots p_r(t)\right)$ where $p_i(t) = \mathbf{Pr}\{\alpha(t) = \alpha_i\}$, and it is the probability that the automaton will select action α_i at time t.

The probability of choosing an action now becomes a function of time. In fact, a VSSA is completely characterized by the method of updating the probability of choosing the actions [3, 5, 12, 13]. This probability distribution over the state of actions is called the probability vector[1]. A single component of this vector will be called an action probability. Note that the components of this vector must sum to unity. The VSSA is in its end state when one of its components is unity .

Definition I.1 A variable structure stochastic automaton (VSSA) is a 4-tuple $\{A, B, T, \mathbf{P}\}$ where A is a finite set of actions, B is a set of outputs from the environment and T: $[0, 1]^r \times B \varnothing [0, 1]^r$ is a learning algorithm such that $T(\mathbf{P}(t), \beta(t)) = \mathbf{P}(t+1)$. $\mathbf{P}(t)$ is the action probability vector such that $\mathbf{P}(t) = \left(p_1(t), p_2(t), \ldots p_r(t)\right)$ with $p_i(t) = \mathbf{Pr}\{\alpha(t) = \alpha_i\}$.

Wherever there is no ambiguity we shall omit the reference to time with the understanding that **P** refers to **P**(t).

[1]In the literature this vector is called the action probability vector. In this paper, for the sake of simplicity, it will often be referred to as the probability vector.

VSSA are analyzed from the point of view that their probability of choosing an action at a given time represents a discrete Markovian process. The probability that an action may be rewarded can remain stationary or non-stationary depending on the environment; VSSA have been developed that are suitable for each situation. Many varieties of VSSA which possess absorbing barriers have been documented [5, 12, 13, 16, 21]. An absorbing barrier is a state that has the property that if the automaton enters this state it is locked there for the rest of time. Ergodic VSSA have also been investigated [12, 13, 19, 21, 32]. Ergodic VSSA can go into and out of any state an unlimited number of times, and their limiting behaviour is independent of their initial state. In non-stationary environments, since the optimal action may change with time, an ergodic VSSA can follow it. In contrast, in stationary environments, automata with absorbing barriers are preferred because they can be made to converge to the optimal action with a probability as close to unity as desired.

Automata can be designed in three varieties of updating schemes: RI, IP, and RP. The letters describe what types of input are required for the probabilities to be updated by the automaton. Their meanings are explained in Table 1.

Table 1
Varieties of Updating Schemes

Symbol	**Full Form**	**Meaning**
RI	Reward-Inaction	Updates probabilities only when automata are rewarded. Penalty responses are ignored.
IP	Inaction-Penalty	Updates probabilities only when automata are penalized. Reward responses are ignored.
RP	Reward-Penalty	Updates probabilities when automata are either
rewarded		or penalized.

For this article B = {0,1} where zero represents a reward and one represents a penalty. Such an environment is called a P-environment in the literature pertaining to learning systems. If the model permits B to be a finite set, $\{0, b_1, b_2, ..b_{n-1}, 1\}$it is called a Q-environment. Finally, if B can be the interval [0,1] it is called is an S-environment.

Definition I.2 A stochastic environment will be defined as a 3-tuple of sets, <*A, B,* **D** >. The sets *A* and *B* are identical to those of the automaton. **D** = [d_1, d_2,…, d_r] is the vector of reward probabilities, where $d_i(t)$ = **Pr{** $\beta(t) = 0 \mid \alpha(t) = \alpha_i$ **}.**

The vector **D** characterizes the environment in the sense that its elements represent the probability that an element of *A* will be rewarded. From the above definition we see that the components of **D** are denoted as $d_i(t)$, where this is the probability that the environment will reward the automaton given that the automaton has chosen action α_i at the instant t. If the probability of any action being rewarded is constant over time, the environment is said to be stationary. Otherwise it is said to be non-stationary.

This article considers only stationary environments. This means that $d_i(t)$ is constant for all t and so the index is dropped and the quantity is denoted as d_i. Note that the dimension of **D** is the same as that of *A* because there is a unique probability of being rewarded for each action in *A*. In order to give the learning automaton a well defined task, it is also assumed that there is a unique maximum component of the vector **D** called d_b where, d_b = $\mathbf{Max}_{1\ i\ r}$ $\{d_i\}$. This action α_b, possessing the highest probability of reward is referred to as the Best Action. The probability of each action being rewarded is unknown to the automaton, and so its task is to decide which action is the best. How the automaton picks the next internal state and the next output is, in essence, the art and the science of designing learning automata.

1.2 Definitions of learning criteria

Intuitively, in the case of learning automata, the task is to find α_b, the Best Action. Let p_b be the probability that the automaton picks α_b.

Definition I.3 A learning automaton is optimal if $p_b(t) \varnothing 1$ as t $\varnothing$, with probability 1.

This definition means that given enough time the learning automaton will eventually discover the right answer. But alas, man is yet to invent a learning automaton that has achieved this [12]. This fact motivates the next definition; if the automaton can't pick the best action with probability unity, then it should be considered good if the probability of picking the best action can be made arbitrarily close to unity. Informally, an automaton is said to be ε-optimal if given enough time and given a large enough internal parameter n, the probability of picking the best action almost all of the time can be made as close to unity as desired. This is formalized in Definition I.4.

Definition I.4 A learning automaton is said to be ε-optimal if the probability of choosing α_b can be made as close to unity as desired. More explicitly, if there is an internal parameter n, such that for all $\varepsilon > 0$, $\delta > 0$, there exists $n_o > 0$, and a $t_o <$ such that **Pr**$[\mid p_b(t) - 1 \mid < \varepsilon] > 1-\delta$ for all t t_o and for all $n > n_o$, then the scheme is ε-optimal [3, 12].[2]

I.3 Applications of LA

Learning algorithms are useful whenever complete knowledge about a stochastic environment is unknown, expensive to obtain or impossible to quantify. Thus they have found applications in various fields including game playing [1,3, 4], pattern recognition [13, 24], and object partitioning [22, 23]. Learning automata are also useful when the environment with which they interact varies with *its* implementation and are thus useful in priority assignments in a queuing system [7, 8], and the routing of telephone calls [14, 15].

II. DISCRETIZED AUTOMATA

The beauty of a discrete learning algorithm is that it does not ignore the limitations of practical implementations; this is used to an advantage. VSSA evolved from fixed structure stochastic automata (FSSA) as an attempt to simplify the analysis of the automata's properties [12]. However, VSSA have a limitation. Implicit in the definition of VSSA is the fact that the probability of choosing an action can be any real number in the interval [0,1]. Rendering this probability space discrete is a general approach for improving VSSA [21, 27]; this is implemented by restricting the probability of choosing an action to only finitely many values from the interval [0,1]. Probability changes are made in jumps, not continuously. In a sense, the discrete VSSA represent a hybrid of FSSA and VSSA. Discrete automata consist of finite sets like FSSA, but they are VSSA because they are characterized by a probability vector which evolves with time. Discrete algorithms are linear if the probability values are equally spaced in the interval [0,1]; otherwise, they are called non-linear [21, 27]. Existing literature [12, 17, 18, 20, 21, 27] uses the term 'discretized' in front of the name of a learning automaton to indicate the discrete version of a continuous VSSA.

II.1 Existing Discrete Automata

Without considering the discrete estimator algorithms described is this article, so far the theoretical properties of seven discrete algorithms have been described [6, 17, 18, 22, 27]. Five of these algorithms are based on the two action linear schemes [3, 12, 13]. All three variants,

[2]In the literature there is another common definition for ε-optimality. A learning automata is said to be ε-optimal if for all $\varepsilon > 0$,

$$\lim_{t\to\infty} \inf p_b(t) > 1 - \varepsilon$$

Intuitively both definitions require that the probability of picking the best action is arbitrarily close to unity. The relationship between the two has not been worked out [21].

L_{RI}, L_{IP}, and L_{RP}, have been discretized. The intent of this section is to give a summary of the properties of these algorithms compared to their continuous counter-parts with a hope that apart from describing the results that are new, the present treatise will also serve as a catalogue of most of the known discretized automata.

The first results concerning discretized learning automata were largely experimental [27]. A discrete version of the two action Linear Reward-Inaction automaton (DL_{RI}) was created and the rate of convergence was compared with the two action Tsetlin and two action Krinsky automata. For various environments, the DL_{RI} scheme was more accurate and reached its end state quicker.

Next, Oommen and Hansen proved that the DL_{RI} is ε-optimal [17]. The latter was also compared to its continuous counterpart [17]. Since the accuracy can be calculated theoretically for both automata, they were given parameters so that they would have the same lower bound for accuracy, in the same environment. The two algorithms were compared to see which would reach this bound first. Experimentally the DL_{RI} reached an accuracy of 99 % quicker than the L_{RI}. Also, for a fixed number of iterations, the DL_{RI} scheme converged to the optimal action more often.

Oommen and Hansen continued work on discrete linear schemes, and proved a version of the Discrete L_{IP} with absorbing barriers to be ε-optimal [17]. This is particularly significant because it was the first linear Inaction-Penalty scheme to be proven ε–optimal. Oommen and Christensen also proved that the Discrete L_{RP} is ε-optimal in certain environments [18]. A version of the DL_{RP} scheme with absorbing barriers was shown to be ε-optimal in **all** environments [18]. These results are significant because their continuous counterparts are **not** ε-optimal in **any** environment! Discrete automata are also ε-optimal outside the linear family of schemes. Oommen proved a non-linear discrete algorithm ε-optimal [21], and Oommen and Christensen discovered a modified discrete L_{RP} scheme that was both ergodic and always ε-optimal [18].

The conditions when linear automata are ε-optimal are presented in Table 2. In the table, the columns represent the three updating strategies: RI, IP and RP. The rows represent the various types of probability spaces encountered: continuous, discrete, and discrete with absorbing barriers. As mentioned earlier, modifications to the two-action Discrete L_{IP}, and the two-action Discrete L_{RP}, which have artificial absorbing barriers have also been investigated. In both these cases, if a component of the probability vector becomes unity or zero it is locked in this state. In this case, the IP and RP schemes are always ε-optimal.

Both the discrete and continuous L_{RI} schemes are absorbing. The key difference between the two algorithms is that the updating method has been changed from being multiplicative to additive. This has subtle effects. For example, consider the case of L_{RI} with $\lambda = 0.5$ and the DL_{RI} with n= 4. Now if, for both automata, action α_1 is chosen and subsequently rewarded, and then action α_2 is chosen and rewarded, the resulting probabilities will be different. At the end of the two iterations, the probability vector of the DL_{RI} scheme returns to its original value. On the other hand, the L_{RI} scheme gives a higher value to the action that has been rewarded most recently. For stationary environments, the DL_{RI} is preferred because, in this sense, it is not biased.

When considering the two action DL_{RI} and DL_{IP} automata, even though the pair of automata are set up differently, their actions can be quite similar. The amazing fact , as seen in Table 2, is that DL_{RI} is ε-optimal in all stochastic environments and DL_{IP} is not ε-optimal in any ! Since their updating mechanism is similar, it seems that the fact that the DL_{RI} is absorbing and DL_{IP} is not accounts for this difference. Indeed this is true, for if DL_{IP} is made absorbing, it becomes ε-optimal [17]. A similar observation can be made for the L_{RP} and the absorbing DL_{RP} [18]. So it *seems to be* a trade off; to get a desirable convergence property in a stationary environment, the discrete linear schemes must be made absorbing.

Table 2
A Comparison of The Asymptotic Properties of Some Linear VSSA

	Reward-Inaction	Inaction Penalty	Reward Penalty
Continuous			
	Algorithm L_{RI}	Algorithm L_{IP}	Algorithm L_{RP}
	Always ε-optimal	*****	$E[p_1] \varnothing c_2/(c_1+c_2)$
Discrete			
	Algorithm DL_{RI}	Algorithm DL_{IP}	Algorithm DL_{RP}
	Always ε-optimal	$E[p_1] \varnothing c_2/(c_1+c_2)$	(i) ε-optimal if $c_b < 0.5$ (ii) Always ε-optimal if responses are filtered[+] .
Discrete Absorbing			
		Algorithm ADL_{IP}	Algorithm ADL_{RP}
		Always ε-optimal	Always ε-optimal

[+] In this case the responses of the environment are not directly fed to the automaton. They are first filtered by a simple coin-tossing experiment. See [18] for the details of this process.

***** Not so easy to characterize.

II.2 Motivation for Discretization

Probably the biggest limitation of learning automata is their slow rate of convergence [12, 28]. By limiting the number of assumptions that learning automata have about the environment, they are a general approach for machine learning. However, this also means that there are fewer properties that can be used to speed up the rate of convergence.

Originally the intent of introducing discrete learning automata was to increase the rate of convergence and to eliminate the assumption that the random number generator could generate real numbers with arbitrary precision [21, 27]. Once the optimal action has been determined, and the probability of selecting that action is close to unity, the discrete automata increase this probability directly, rather than approach the value unity asymptotically.

If the quantity p_1 goes directly from 0.98 to 1.0, this will change the expected probability of the LA getting a reward from $(0.98\ d_1 + 0.02\ d_2)$ to d_1. In the worst case this corresponds to at most a 2 % change in the expected probability of being rewarded. If the jump to the end state reduces the number of iterations by 50%, the trade-off may be worth while. By making the probability space discrete, a minimum step size is obtained. If the automaton is close to an end state, the minimum step size forces it to this state with just a few more favourable responses.

The central issue from a theoretical point of view is that the properties of a Markov process can change if the probability of choosing an action is restricted to a finite subset of [0,1]. For example, a continuous space will have recurrent states, but a finite space will only have positive recurrent states [25]. As well, discrete Markov processes have properties that are not true for general Markov processes [25]. Round off error will cause the automaton that approaches its end point asymptotically to artificially reach its end point [18, 21]. Also, the proofs of convergence in continuous spaces may not be applicable to a finite state machine. This point is demonstrated by the fact that, so far, the existing proofs of convergence for discrete algorithms are significantly different from the proofs of their continuous counterparts (compare [17] and [21] to the methods used in [3,5,12]).

Another benefit of discretizing the probability of choosing an action is that it reduces the requirements on the system's random number generators [18, 21]. This is important since VSSA use a random number generator in its implementation [27]. In theory, it is assumed that any real value in [0, 1] can be obtained from the machine; in practice, only a finite number of these values are available.

Finally, and far from being unimportant are the considerations of implementation and representation. Discrete versions lead, quite naturally, to the use of integers for keeping track of how many multiples of $^1/_n$ the action probabilities are. While the above consideration frequently increases the rate of convergence measured in terms of the number of iterations, a discrete algorithm also has the benefit of reducing the time measured in terms of the clock cycles that a microprocessor would take to do **each** iteration of the task. It also reduces the amount of memory needed. Typically addition is quicker than multiplication on a digital computer, and the amount of memory used for a floating point number is usually more than that required for an integer. In the schemes that have been discretized so far, whereas the continuous versions update their probability vectors via multiplication, the discretized counterparts achieve this with addition and subtraction. Thus, in terms of both time and space, discrete algorithms seem to be superior.

III. THE RATIONALE BEHIND ESTIMATOR ALGORITHMS

A recent approach to improving the rate of convergence of LA was introduced by the pioneer, Thathachar, and his student Sastry using the so-called estimator algorithms [26, 28-31]. Whereas all VSSA have a probability vector, where the i^{th} component represents the probability of choosing the i^{th} action, Estimator Algorithms are characterized by also having an estimate of the probability of each action being rewarded. This will be referred to as the estimate vector.

Typically, non-estimator algorithms update the probability of choosing an action based directly on the feedback from the environment. Estimator Algorithms are different. The probability vector is updated based on both the estimate vector and the current response from the environment. Thus, for Estimator Algorithms, even if a particular action was rewarded, it may happen that the probability of choosing another action is increased ! As Thathachar and Sastry point out, this is a novel feature of these algorithms [28]. The extra computational complexity of having an estimate vector pays off handsomely ; the rate of convergence is greatly increased when compared to traditional algorithms such as the L_{RI} scheme [26, 28, 31].

III.1 The Pursuit Algorithm

The Pursuit Algorithm is a special case of a general estimator algorithm [11, 29, 31]. This algorithm is characterized by the fact that it pursues what it reckons to be the optimal action. This involves three steps [31]. The first step is to pick the action $\alpha(t)$ based on the probability distribution $\mathbf{P}(t)$. If the automaton is rewarded the second step is to increase the component of $\mathbf{P}(t)$ whose current estimate of reward is maximal. Finally, the third step updates the running estimates of the probability of being rewarded. To do this, two more variables are introduced: Z_i is the number of times the i^{th} action has been chosen up to time t; and W_i is the number of times the i^{th} action has been rewarded up to time t. Observe that for all i, the i^{th} component of the estimate vector, $d'_i(t)$, equals W_i / Z_i. Explicitly the algorithm is stated as follows:

```
ALGORITHM Pursuit
Parameters
        $\lambda$:      the speed of learning parameter where $0 < \lambda < 1$.
        m:  index of the maximal component of $\mathbf{D}'(t)$, $d'_m(t)=\mathrm{Max}_i\{d'_i(t)\}$.
        $\mathbf{e}_m$:     the unit r-vector with 1 in the m$^{th}$ coordinate.
        $W_i(t)$:  the number of times the i$^{th}$ action has been rewarded up to time t, with 1 i
r.
        $Z_i(t)$:  the number of times the i$^{th}$ action has been selected up to time t, with 1  i
r.
Method
    Initialize $p_i(0) = 1/r$ for 1  i   r
    Initialize $\mathbf{D}'(0)$ by picking each action a small number of times.
    Repeat
        Step 1 :  At time t pick $\alpha(t)$ according to probability distribution $\mathbf{P}(t)$.
        Step 2 :  Update $\mathbf{P}(t)$ according to the following:
            $\mathbf{P}(t+1) = \mathbf{P}(t)$                          if $\beta(t) = 1$
                      $= (1 - \lambda)\, \mathbf{P}(t) + \lambda\, \mathbf{e}_m$          if $\beta(t) = 0$

        Step 3 :  Update $\mathbf{D}'(t)$ according to the following:
            If $\alpha(t) = \alpha_j$,
                $W_j(t+1) = W_j(t) + (1 - \beta(t))$
                $Z_j(t+1) = Z_j(t) + 1$
                $d'_j(t+1) = W_j(t+1) / Z_j(t+1)$
            For all i j
                $W_i(t+1) = W_i(t)$
                $Z_i(t+1) = Z_i(t)$
                $d'_i(t+1) = d'_i(t)$.
    EndRepeat
END ALGORITHM Pursuit
```

The Pursuit Algorithm is similar in design to the L_{RI} algorithm, except that whereas the L_{RI} algorithm moves **P**(t) in the direction of the most recently rewarded action, the Pursuit Algorithm moves **P**(t) in the direction of the action which possesses the highest estimate of reward. Both algorithms approach their end points (when **P**(t) is a unit vector) asymptotically. This is because the action probability to which the scheme is converging to is increased by a quantity proportional to the sum of the remaining action probabilities.

Thathachar and Sastry [31] did experimental tests to compare the two automata. The two algorithms were compared to determine the number of iterations necessary to achieve the same level of accuracy. The Pursuit Algorithm was five to seven times faster than the L_{RI} algorithm. Like the L_{RI} scheme, the Pursuit Algorithm has been shown to be ε-optimal [31].

III.2 The TS Estimator Algorithm[3]

The TS Estimator (TSE) Algorithm is a far more complex scheme. Given that the α_i has just been rewarded, this algorithm treats probability components with a estimate of reward greater than d'_i differently from those with a value lower than d'_i. In order to facilitate this, Thathachar and Sastry use an indicator function $S_{ij}(t)$ which is zero unless d'_i is bigger than d'_j, in which case it is unity. Basically the probabilities are updated as a function of both the reward estimates **D'**(t), and the action probability vector **P**(t), as seen in (3.1a) and (3.1b).

This algorithm is formally described as below. The updating of **P**(t+1) depends indirectly on the response from the environment. Feedback from the environment changes the value of the components **D'**(t). This in turn affects the value of a user-defined function $f\big(d'_i(t) - d'_j(t)\big)$ and the value of $S_{ij}(t)$. The function $S_{ij}(t)$ determines whether the term $\lambda.f\big(d'_i(t) - d'_j(t)\big)$ is multiplied by $p_i(1-p_j)/(r-1)$ or by p_j.

ALGORITHM TSE
Parameters

```
        $\lambda$, m, $e_m$, $W_i(t)$, $Z_i(t)$ are the same as in the Pursuit Algorithm.
        $S_{ij}(t)$ : An indicator function, where,
            $S_{ij}(t)$   = 1    if $d'_i(t) > d'_j(t)$
                          = 0    if $d'_i(t) \quad d'_j(t)$.
        f : a monotonic, increasing function[4], where f:[ 0, 1] ∅ [ 0, 1], f(0)=0 and f(1)=1.
Method
    Initialize $p_i(0) = 1/r$ for $1 \quad i \quad r$
    Initialize D'(0) by picking each action a small number of times.
    Repeat
        Step 1 :   At time t pick $\alpha(t)$ according to probability distribution P(t).
        Step 2 :   Update P(t) according to the following. Let $\alpha(t) = \alpha_i$.
```

For all j, such that j   i,

$$p_j(t+1) = p_j(t) - \lambda\Big[f\big(d'_i(t)-d'_j(t)\big)\Big(S_{ij}(t)\,p_j(t) + S_{ji}(t)\,\frac{(1-p_j(t))\,p_i(t)}{r-1}\Big)\Big] \quad (3.1a)$$

[3]Thathachar and Sastry refer to this algorithm as just an estimator algorithm. However this paper must distinguish it from the other estimator algorithms, and so we will call it the TS Estimator (TSE) algorithm.

[4]The typical family of functions that are used here are $f(x) = x^n$, where n is one of {. . . 1/4, 1/3, 1/2, 1, 2, 3, 4, .}.

$$p_i(t+1) = p_i(t) + \lambda \sum_{j \quad i} \left[f\left(d'_i(t) - d'_j(t)\right) \left(S_{ij}(t)\, p_j(t) + S_{ji}(t) \; \frac{(1-p_j(t))\, p_i(t)}{r-1} \right) \right] \tag{3.1b}$$

```
        Step 3 :    Same as in the Pursuit Algorithm.
    EndRepeat
END ALGORITHM TSE
```

In order to describe the TSE Algorithm, the S_{ij} notation will be temporarily dropped. This breaks down the updating rule into three cases. If the ith action is rewarded, then all actions with estimates bigger than i will be updated according to the following rule:

$$p_j(t + 1) = p_j(t) + \lambda \left[f\left(d'_i(t) - d'_j(t)\right) \left(\frac{(1-p_j(t))\, p_i(t)}{r-1} \right) \right] \tag{3.2}$$

All actions with estimates of reward less than action i will be updated according to the following:

$$p_j(t + 1) = p_j(t) - \lambda \left[f\left(d'_i(t) - d'_j(t)\right) \left(p_j(t) \right) \right] \tag{3.3}$$

The component that is calculated last is $p_i(t + 1)$, which is given the value so that the sum of all the probabilities in the vector **P**(t+1) is unity. It is worth noting that if for all x, f(x) = 1 then (3.3) reduces to the Pursuit algorithm above.

If $d'_i(t)$ is less than $d'_j(t)$ then the value of $f\{d'_i(t) - d'_j(t)\}$ is negative because f is monotonic and increasing. So the value of $p_j(t + 1)$ is bigger than $p_j(t)$ in (3.2). In other words, if there are actions with estimates bigger than the ith action's estimate, their probability of being chosen will increase. This is just as Thathachar and Sastry point out [30]: though one action is rewarded, other actions may increase their probability of being chosen. As a matter of fact, if the ith action has the smallest estimate of reward probability, the value of all the other action probabilities will increase and so the value of p_i will actually decrease even though α_i was rewarded ! This is the reason why the $p_i(t) / (r - 1)$ term was included in (3.2). Even if the ith action has the smallest estimate of being rewarded, the total increase of all the other $p_j(t)$, $j \quad i$, will never be bigger than the value of $p_i(t)$.

The next consideration is the convergence of the algorithm. The $(1 - p_j(t))$ factor in (3.2) and the $p_j(t)$ factor in (3.3) ensure that the change in these components is proportional to their original value. As the values get closer to their end states (zero or unity) the magnitude of change of the probability vector gets smaller. Assume that α_i has the maximal estimate of reward probability. Then all the other components, $p_j(t)$, $j \quad i$, will be updated according to (3.3). Since α_i has the maximal estimate for the reward probability, the term $[1 - \lambda \; f(d'_i(t) - d'_j(t))]$ is strictly less than one. Thus, if the algorithm has been running for a while, both $d'_i(t)$ and $d'_j(t)$ will remain approximately constant, and so $p_j(t)$ will decrease asymptotically. As with the Pursuit algorithm, for practical implementations the scheme will eventually round off to an end state. But in the case of machines with finite accuracy, if the end state action is penalized enough so that its estimate of reward probability is no longer maximal, the sign of $d'_i(t) - d'_j(t)$ will change and the scheme can move away from that end state.

This algorithm has also been shown to be ε-optimal [30]. Because of the unique possibility of decreasing an action that has just been rewarded, the TSE Algorithm has no analogous non-estimator counterpart. Simulation comparisons with the L_{RI} were done by Thathachar and Sastry. The environment and set up of the experiments were identical to those of the Pursuit algorithm. For a given level of accuracy the TSE Algorithm converges an order of magnitude quicker [30].

IV. DISCRETE ESTIMATOR ALGORITHMS

IV.1 Motivation

We shall now combine the two previous approaches of catalyzing the convergence of VSSA to create a class of algorithms called Discrete Estimator Algorithms (DEA). The probability vector will be restricted to only finitely many values to reflect the constraints of discrete algorithms, and estimate vectors will be used to update the probability vector in order to utilize the benefits of estimator algorithms. Throughout this chapter, n will be a resolution parameter and the interval [0,1] is subdivided into a number of intervals proportional to n.

Discrete versions of the Pursuit algorithm[5] and the TSE Algorithm will be considered. Both algorithms are ε-optimal. Indeed, we shall present an entire family of DEAs. Members of this family will be shown to be ε-optimal by deriving two sufficient conditions required for the ε-optimality. We conjecture that the necessity of these conditions too.

The first property that a DEA must possess is that it must implicitly specify an upper bound on the amount any action probability can decrease during a single iteration. Using the notation that r is the number of actions and n is a resolution parameter, this property, called the *Property of Moderation*, is stated as follows:

Property 1: A DEA with r actions and a resolution parameter n is said to possess the *Property of Moderation* if the maximum magnitude by which an action probability can decrease per iteration is bounded by $^1/_{r\,n}$.

[5]The discretized version of the Pursuit Algorithm can be found as an algorithm in its own right in [6,30]. However, in this paper we would like to emphasize that it is only a member of the family of discretized estimator algorithms which satisfy the sufficient conditions given in this section.

The next property is called the *Monotone Property*. If the estimate of reward of an action, say α_m, remains the maximum estimate subsequent to a certain point of time, then, we say that a DEA possesses the *Monotone Property* if it steadily increases the probability of choosing α_m to unity.

Property 2: Suppose there exists an index m and a time instant $t_o <$, such that $d'_m(t) > d'_j(t)$ for all j such that j m and all t t $_o$. A DEA is said to possess the *Monotone Property* if there exists an integer n_o such that for all resolution parameters $n > n_o$, $p_m(t)$ ∅ 1 with probability one as t ∅ .

We now present two DEAs possessing the moderation and monotone properties.

IV.2 The Discrete Pursuit Algorithm

The Discrete Pursuit Algorithm (DPA) mimics the strategy followed by the continuous Pursuit Algorithm, except that the probability changes are made in discrete steps. Thus terms involving multiplication by λ are replaced by the addition or subtraction of the smallest step size.

Thus the algorithm works in three steps just like the continuous Pursuit Algorithm. The difference is in the second step. For the DPA, the components of the probability vector are increased by integral multiples of the smallest step size Δ, where $\Delta = 1/rn$. If the automaton has been rewarded and has not converged, then all the non-zero action probabilities are decreased by Δ except the one with the highest estimate of the reward probability. This action probability is increased by the appropriate amount to keep the sum of the components of the vector equal to unity. Explicitly, the algorithm is as described below.

```
ALGORITHM DPA
Parameters
        m, $W_i(t)$, $Z_i(t)$ are the same as in the Pursuit Algorithm.
        $\Delta = 1/rn$ is the smallest step size.
Method
    Initialize $p_i(0) = 1/r$ for 1  i   r
    Initialize D'(0) by picking each action a small number of times.
    Repeat
      Step 1 :   At time t pick $\alpha(t)$ according to probability distribution P(t).
      Step 2:    Let $\alpha(t) = \alpha_k$. Update P(t) based on the components of D'(t).
         If $\beta(t) = 0$ and $p_m(t)$   1
            $p_j(t+1) = \mathbf{Max\{}\, p_j(t) - \Delta, 0\, \mathbf{\}}$     for all j   m
            $p_m(t+1) = 1 - \sum_{j \quad m} p_j(t+1)$
```

```
                Else
                    $p_j(t+1)= p_j(t)$    for all 1   j   r
(4.1)
            Step 3 :    Same as in the Pursuit Algorithm.
        **EndRepeat**
**END ALGORITHM DPA**
```

The fact that this algorithm has the two necessary properties will now be proved.

Lemma 4.1 The DPA possess the moderation property.

Proof: The result is true since, in the worst case, any component of **P**(t) can decrease by at most $\Delta={}^1/_{rn}$.

Lemma 4.2 The DPA possess the monotone property.

Proof : Suppose for the DPA, there exists an index m and a time instant $t_o <$ such that $d'_m(t) > d'_j(t)$ for all j such that j m and all t   t $_o$. Then we have to prove that there exists an integer n_o such that for all resolution parameters $n > n_o$, $p_m(t) \varnothing 1$ with probability one as $t \varnothing$.

Consider the sequence of random variables $\{p_m(t)\}_{t\ t\ o}$ satisfying $\sup_{t\ 0}$ $\mathbf{E}[\left| p_m(t) \right|] <$. We shall show that this sequence constitutes a submartingale. If that is the case, we shall make use of the martingale convergence theorem [2] which states that the sequence converges with probability one, i.e.

$$\mathbf{Pr}\left\{ \lim_{t \varnothing} \quad p_m(t) = p_m \right\} = 1$$

We shall first prove that this sequence $\{p_m(t)\}_{t\ t\ o}$ is a submartingale. For the DPA, we know that if m satisfies

$$d'_m = \mathbf{Max}_i \{ d'_i(k) \}$$

then, $d'_m(t) > d'_j(t)$ for all j m and all t   t $_o$. Therefore, for all $t > t_o$,

$$\begin{aligned} p_m(t+1) &= p_m(t) && \text{if } \beta(t) = 1 && (\text{ that is w.p. } 1 - d_m\ ) \\ &= 1 - \sum_{j\ m} \mathbf{Max}(\ p_j(t) - \Delta, 0) && \text{if } \beta(t) = 0 && (\text{ that is w.p. } d_m\ ). \end{aligned}$$

If $p_m(t) = 1$ then the absorbing property of the algorithm trivially proves the result.

Of course, there is the possibility that the algorithm has already converged to an action α_j where j m. To ensure that this has not happened, we assume that the resolution parameter is large enough so that the algorithm has not converged by the time t_o. Assuming the algorithm has not converged, there exists at least one non-zero component of **P**(t) beside $p_m(t)$, say $p_k(t)$, and hence we assert that

$$\mathbf{Max}\ (p_k(t) - \Delta, 0) < p_k(t).$$

Since **P**(t) is a probability vector $p_m(t) = 1 - \sum_{j\ m} p_j(t)$ and so,

$$1 - \sum_{j\ m} \mathbf{Max}(\ p_j(t) - \Delta, 0) \quad > p_m(t).$$

As long as there is at least one non-zero component $p_k(t)$ (where k m), it is clear that we can decrement $p_k(t)$ and hence increment $p_m(t)$ by Δ. Hence,

$$p_m(t+1) \ = \ p_m(t) + \chi\Delta,$$

where is χΔ is an integral multiple of Δ, χ is bounded by 0 and r, and Δ is the smallest step size. Therefore we write,

$$\mathbf{E}[p_m(t+1) \big| \mathbf{P}(t), \mathbf{D'}(t), p_m(t) \quad 1\ ] \ = d_m \{ p_m(t) + \chi\Delta \} + (1 - d_m)\{p_m(t)\}$$
$$= p_m(t) \ + \ d_m\chi\Delta$$

Since the above two terms have an upper bound of unity,

$\mathbf{E}\ [\ p_m(t+1) \ \big| \mathbf{P}(t), \mathbf{D'}(t), p_m(t) \quad 1]$ is bounded. Hence

$\sup_{t\ 0} \ \mathbf{E}[\ \big| p_m(t+1) \big| \mathbf{P}(t), \mathbf{D'}(t), p_m(t) \quad 1] <$. Thus,

$\mathbf{E}\ [\ p_m(t+1) - p_m(t) \big| \mathbf{P}(t), \mathbf{D'}(t)\ ] = d_m\chi\Delta \quad 0$ for all t t_o,

implying that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [2], $\{p_m(t)\}$ converges and so as t∅ ,

$\mathbf{E}\ [\ p_m(t+1) - p_m(t) \big| \mathbf{P}(t), \mathbf{D'}(t)\ ] \varnothing 0$ w.p. 1

implying that $d_m\chi\Delta \ \varnothing 0$ w.p. 1. This in turn implies that $\chi \varnothing 0$ w.p. 1, and consequently that

$\sum_{j\ m} \mathbf{Max}(\ p_j(t) - \Delta, 0) \quad \varnothing 0$ w.p. 1. Hence $p_m(t) \varnothing 1$ w.p. 1, and the result is proved.

IV.3 The Discrete TSE Algorithm

Like its continuous predecessor, the Discrete TSE (DTSE) Algorithm is a far more complex scheme. However it serves to illustrate the power of the general proof of convergence that follows in the next section. Its design is merely a compromise between the necessity of having the algorithm possess the moderation and monotone properties while possessing as many qualities of the continuous algorithm as possible.

The discrete scheme is a two parameter system. The smallest step size for this algorithm is $\Delta = {}^1/_{rn\theta}$. Here θ represent the largest integer multiple of Δ that any one component of the probability vector can decrease by in one iteration. The continuous updating rule has three factors:

1) λ

2) $f\big(d'_i(t)-d'_j(t)\big)$

3) $\left(S_{ij}(t)\, p_i(t) + S_{ji}(t)\, \frac{(1-p_j(t))\, p_i(t)}{r-1} \right)$

The modification to each one will be dealt with in turn. The first term, λ, represents the maximum that the continuous probability component can change, and so this has been replaced by θ, an integer. The $f\big(d'_i(t)-d'_j(t)\big)$ term remains intact because it is a feature of the TSE Algorithm to base the magnitude of the increase on the difference in estimates of reward. In the third factor the terms $p_i(t)$ and $\big(1-p_j(t)\big)\, p_i(t)$ have been dropped completely. This is in the spirit of the general goal of discrete algorithms which is that of not having the algorithm approach its end point asymptotically. So the third term becomes $\left(S_{ij}(t) + S_{ji}(t)\, \frac{1}{r-1}\right)$. The way these terms are brought together is:

$$\theta\, f\big(d'_i(t)-d'_j(t)\big) \left(S_{ij}(t) + S_{ji}(t)\, \frac{1}{r-1} \right)$$

The above term when rounded up, determines the multiple of Δ that will be used to update the probability vector. Given that α_i has just been rewarded, this algorithm treats probability components with a higher estimate of reward than d'_i differently from those with a lower estimate than d'_i. To facilitate this the indicator function $S_{ij}(t)$ has been used just as in the continuous case.

We now introduce two special functions. The first is Rnd() which rounds up its parameter so that its value is always an integer. The second function is Check. Given three parameters, p_i, p_j, and x, Check$\big(p_i, p_j, x\big)$ calculates the largest integer multiple of Δ, between 1 and x, that can be added to p_i and subtracted from p_j while simultaneously preserving the fact that p_i and p_j are bounded between zero and one. It is important to note that the first component that is updated is the maximal one, so as to guarantee that this value will always increase for each iteration. This is necessary to satisfy the monotone property. As a final note we would like to state that this scheme is "operationally" different than the traditional estimator and non-estimator schemes because every single change of p_j which is reflected in the magnitude of p_i must be explicitly verified so as to ensure that **P** is a probability vector. This is done in (4.2a) and (4.2b). By comparing these with (3.1a) and (3.1b) we can observe that this verification does not increase the computational complexity of the scheme. The algorithm is formally described below.

ALGORITHM DTSE

Parameters

m, $S_{ij}(t)$, f, $W_i(t)$, $Z_i(t)$ are the same as in the TSE Algorithm.

$\Delta = {}^1/_{r n \theta}$, with θ (an integer) being the maximum any component can change.

Rnd(x) rounds up x to one of $\{-\theta, -\theta+1, -\theta+2, \ldots \theta-1, \theta\}$.

```
    Check$\left(p_i(t), p_j(t), x\right)$ returns the largest integer less than or equal to x such that
    $0 \quad p_i(t)+x\Delta,\ p_j(t)-x\Delta \quad 1$.
Method
        Initialize $p_i(0) = 1/r$ for $1 \ \ i \ \ r$
        Initialize $\mathbf{D}'(0)$ by picking each action a small number of times.
        Repeat
            Step 1 :   At time t pick $\alpha(t)$ according to probability distribution $\mathbf{P}(t)$.
            Step 2 :   Let $\alpha(t) = \alpha_i$. Update $\mathbf{P}(t)$ according to the following:
                For each action i, starting with m Do
                    change = $\mathrm{Rnd}\left(\theta\, f\left(d'_i(t)-d'_j(t)\right)\left(S_{ij}(t) + S_{ji}(t)\ \frac{1}{r-1}\right)\right)$
                    $p_j(t+1) = p_j(t) - \Delta\, \mathrm{Check}\left(p_i(t), p_j(t), \text{change}\right)$     (4.2a)
                    $p_i(t+1) = p_i(t) + \Delta\, \mathrm{Check}\left(p_i(t), p_j(t), \text{change}\right)$     (4.2b)
                EndFor

            Step 3 :   Same as in the Pursuit Algorithm.
        EndRepeat
END ALGORITHM DTSE
```

We now prove the properties of the DTSE.

Lemma 4.3 The DTSE Algorithm possesses the moderation property**.**

Proof : We need to show that the magnitude by which any action probability can decrease at any one iteration of the algorithms is bounded by $^1/_{r\,n}$.

There are two possible worst cases:

i) If any $p_j(t) < p_i(t)$ then it will decrease by

$$\Delta\, \mathrm{Rnd}\left(f\left(1\right)\theta\right) = \frac{1}{rn\theta}\ \mathrm{Rnd}\left(\theta\right) = \frac{1}{rn}\ .$$

ii) If any $p_j(t) > p_i(t)$ then it will get decreased r-1 times by the amount

$\Delta\, \mathrm{Rnd}\left(f\left(1\right)\ \frac{\theta}{r-1}\ \right) = \frac{r-1}{rn\theta}\ \mathrm{Rnd}\left(\frac{\theta}{r-1}\right) = \frac{1}{rn}$, and the result is proved.

Lemma 4.4 The DTSE Algorithm possesses the monotone property.

Proof : Suppose there exists an index m and a time instant $t_o <$ such that $d'_m(t) > d'_j(t)$ for all j m and all t t_{o}. Then we need to show that there exists an integer n_o such that for all resolution parameters $n > n_o$, $p_m(t) \emptyset 1$ with probability one as t $\emptyset$.

As in the case of the DPA consider the sequence of random variables $\{p_m(t)\}_{t\ t_o}$ satisfying $\sup_{t\ 0}$ $\mathbf{E}[|\ p_m(t)\ |] <$. We shall first show that this sequence constitutes a submartingale. For the DTSE Algorithm, we know that if m satisfies $d'_m = \mathbf{Max}_i\ \{ d'_i(k) \}$, then,

$$d'_m(t) > d'_j(t) \text{ for all j m and all t } \ t_o.$$

Therefore, for all $t > t_o$, $S_{mj}(t) = 0$, and so

$$p_m(t+1) = p_m(t) \qquad \text{if } \beta(t) = 1 \qquad \text{(that is w.p. 1- } d_m)$$

$$= 1 - \Delta \sum_{j\ m} \text{Rnd}\Big(f\big(d'_m(t)-d'_j(t)\big)\ \theta\Big) \qquad \text{if } \beta(t) = 0 \qquad \text{(that is w.p. } d_m).$$

If $p_m(t) = 1$ then the absorbing property of the algorithm trivially proves the result.

Of course, there is the possibility that the algorithm has already converged to an action α_j where j m. To ensure that this has not happened, as in the case of the DPA, we assume that the resolution parameter is large enough so that the algorithm has not converged by t_o. Assuming that the algorithm has not converged, there exist at least two non-zero components of $\mathbf{P}(t)$, say $p_k(t)$ and $p_m(t)$. By the assumption of the property $d'_m(t) > d'_k(t)$, and hence we assert that

$$p_k(t+1) = p_k(t) - \frac{1}{rn\theta}\ \text{Rnd}\Big(f\big(d'_m(t)-d'_k(t)\big)\ \theta\Big) < p_k(t) - \Delta.$$

Since $\mathbf{P}(t)$ is a probability vector if $p_k(t)$ decreases by at least Δ then $p_m(t)$ must increase by at least Δ. So, $p_m(t+1) > p_m(t) + \Delta$. Therefore,

$$\mathbf{E}\,[p_m(t+1)\,|\,\mathbf{P}(t), \mathbf{D}'(t), p_m(t)\ \ 1] \ > \ d_m\,\{p_m(t) + \Delta\} + (1- d_m)\,\{p_m(t)\}$$
$$= p_m(t) + d_m\Delta$$

Since the above two terms have an upper bound of unity,

$$\mathbf{E}\,[\,p_m(t+1)\ |\,\mathbf{P}(t), D'(t), p_m(t)\ \ 1] \text{ is bounded.}$$

Hence $\sup_{t\ 0}$ $\mathbf{E}\,[\,|\,p_m(t+1)\,|\,\mathbf{P}(t), \mathbf{D}'(t), p_m(t)\ \ 1] <$. Thus,

$$\mathbf{E}\,[\,p_m(t+1) - p_m(t)\,|\,\mathbf{P}(t), \mathbf{D}'(t)\,] = d_m\Delta \quad 0 \text{ for all t } \ t_o,$$

implying that $p_m(t)$ is a submartingale.

By the submartingale convergence theorem [2], $\{p_m(t)\}$ converges and so as t∅ ,

$$\mathbf{E}\,[\,p_m(t+1) - p_m(t)\,|\,\mathbf{P}(t), \mathbf{D}'(t)] \varnothing 0 \text{ w.p. 1}$$

But any change, while not in an end state is bounded by $d_m\Delta$. Hence the automaton must ultimately terminate in an end state. Hence $p_m(t) \varnothing 1$ w.p. 1, and the lemma is proved.

We now study the convergence of any scheme possessing the moderation and monotone properties.

IV.4 Proof Of Convergence

The asymptotic property of DEAs will now be proved. The proofs of Theorems 4.1 and 4.2 are analogous to that of the TSE Algorithm [30] with the appropriate modifications for the change from a continuous to a discrete space. From a broad perspective, there are three differences of note. The first is trivial. The continuous pursuit algorithms have a learning parameter λ, and as λ tends to zero the algorithm takes a longer time to converge. The discrete version has the integer parameter n which is bounded by one and infinity. As n approaches infinity the algorithm takes a longer time to converge. But, in one sense these two parameters are interchangeable. The second difference is that, whereas the parameter of the continuous schemes vary continuously, the parameter of the discrete scheme varies in a discrete manner. Thus the limiting arguments have to be used in somewhat different ways. The final difference is with the proof of Theorem 4.1. Unlike [30], this proof is achieved without the erroneous assumption that the random vectors **P**(t) and **P**(t+1) are independent[6]. With these modifications accounted for, Theorem 4.2 can be proved using the same method as its continuous counterpart [30].

We shall now prove the convergence of the scheme. Let DEA represent a discrete estimator algorithm with the moderation and monotone properties. Indeed, in every stationary random environment, a DEA is ε-optimal. This will be proved by showing that given $\varepsilon > 0$ and $\delta > 0$, there exist an $n_o > 0$ and a $t_o <$ such that for all time t t $_o$ and for any resolution parameter $n > n_o$ the following is true:

$$\mathbf{Pr[} \mid 1 - p_b(t) \mid < \varepsilon \mathbf{]} > 1 - \delta.$$

Note that in the above p_b refers to the probability of choosing the best action. Our proof is motivated by the fact that the algorithms have been set up so that each action can be sampled an arbitrary number of times by suitably choosing the internal parameter n. Thus one can obtain arbitrarily accurate estimates of the reward probabilities, which in turn can yield sufficient discrimination among the actions. We shall first prove that any DEA has this property.

Theorem 4.1

For each action α_i, assume $p_i(0)$ 0. Then for any given constants $\delta > 0$ and M < , there exist $n_o <$ and $t_o <$ such that under the DEA, for all learning parameters $n > n_o$ and all time $t > t_o$, the probability **Pr**{each action chosen more than M times at time t } 1 - δ.

Proof :

Define the random variable Y_t^i as the number of times the i th action was chosen up to time t. Then we must prove that

[6]We are grateful to Prof. Dixon, from the Department of Mathematics at Carleton University for pointing out that in general, this assumption is false.

$$\mathbf{Pr}\{Y_t^i \; > M \} \quad 1 - \quad \delta. \tag{4.3}$$

Clearly (4.3) is equivalent to $\mathbf{Pr}\{ Y_t^i \quad M \} < \quad \delta.$ (4.4)

The events $\{Y_t^i = k\}$ and $\{Y_t^i = j\}$ are mutually exclusive for j   k, and so (4.4) yields (4.5).

$$\sum_{k=1}^{M} \mathbf{Pr} \{ Y_t^i = k \} < \delta. \tag{4.5}$$

For any iteration of the algorithm, $\mathbf{Pr}\{\alpha_i$ is chosen $\}$ 1. As well, by the moderation property, during any of the first t iterations of the algorithm :

$$\mathbf{Pr}\{\alpha_i \text{ is not chosen }\} \quad (1- p\ \ _i(0) + {}^t/_{rn}).$$

Using these two upper bounds, the probability that action α_i is chosen at most M times among t choices has the following upper bound :

$$\mathbf{Pr}\{Y_t^i \quad M\} \quad < \quad \sum_{k=1}^{M} C(t,k)(\ 1\ )^{k} (\ 1- p_i(0) + {}^t/_{rn})^{t-k} \tag{4.6}$$

However, to make a sum of M terms less than δ, is is sufficient to make each element of the sum less that ${}^\delta/_M$. So we will pick an arbitrary term, say when k = m. It suffices to show that the mth term is less that ${}^\delta/_M$ or equivalently that M multiplied by the mth term is less that δ. Hence we must show that M $C(t,m)(\ 1\ )^m (\ 1- p_i(0) + {}^t/_{rn})^{t-m}$ can be made to be bounded by δ.

First of all, C(t,m)   t ^{m}. This leads us having to show that :

$$M\, t^m\ (\ 1- p_i(0) + {}^t/_{rn})^{t-m} < \ \delta.$$

Now in order to get the R.H.S. of this term less then δ as t increases, the $(\ 1 - p_i(0) + {}^t/_{rn}\ )$ term must be strictly less than unity. In order to guarantee this, we bound the value of n with respect to t in such a way that $(1- p_i(0) + {}^t/_{rn}) < 1$. We do this by requiring that $n > \dfrac{t}{rp_i(0)}$.

$$\text{Let, } n = \frac{2t}{rp_i(0)} \tag{4.7}$$

With this value of n (4.6) simplifies to $\mathbf{Pr}\{ Y_t^i \quad M \} < M\, t^{\ m}\ \psi^{t-m}$, where $\psi = \ 1 - {}^1/_2\, p_i(0)$ and $0 < \ \psi < \ 1$.

It remains to evaluate, $\lim\limits_{t\,\varnothing}\ M\, t^m\, \psi^{m-t}$. This is equivalent to,

$$M \lim_{t\,\varnothing} \quad \frac{t^m}{(1/\psi)^{t-m}} \qquad \text{with } n = \frac{2t}{rp_i(0)} \tag{4.8}$$

Now (4.8) is in an indeterminate form. By using l'Hopital's rule m times the following is obtained:

$$M \lim_{t \varnothing} \frac{m!}{(\ln 1/\psi)^{m}(1/\psi)^{t-m}} = 0, \qquad \text{with } n = \frac{2t}{rp_i(0)}$$

This is, (4.8) has a limit of zero as t and n tends towards infinity, when condition (4.7) is satisfied. Hence by virtue of the fact that the limits exists, for every action α_i there is a t(i) such that for all t > t(i) (4.6) is less that δ when condition (4.7) is satisfied. Now set $n_{(i)} = \frac{2t(i)}{rp_i(0)}$. It remains to be shown that (4.6) is satisfied for all n > n(i) and for all t > t(i). This is trivial because as n increases the L.H.S. of (4.6) is monotonically decreasing, and so the inequality (4.6) is preserved. Also, for any t > t(i), no matter how often action α_i is chosen in the interval from t(i) to t, by the properties of probability: **Pr** { ($Y^{i}_{t(i)}$ M) and (any other event) } **Pr** { ($Y^{i}_{t(i)}$ M)}. Thus in this case too, the inequality (4.6) is preserved. Thus for any action α_i:

Pr { ($Y^{i}_{t(i)}$ M)} < δ whenever t > t(i) and n > $n_{(i)}$.

Since, we can repeat this argument for all the actions, we can define t_o and n_o as follows: t_o = $\text{Max}_{1\ i\ r}$ { t(i) } and n_o = $\text{Max}_{1\ i\ r}$ { $n_{(i)}$ }. Thus for all i we have shown that for all t > t_o and all n > n_o, the quantity **Pr**{ Y^{i}_{t} < M } < δ, and the theorem is proved.

With this property established we now prove the central result of this paper.

Theorem 4.2

In every stationary random environment, the family of DEA are ε-optimal.

Proof : We need to show that given $\varepsilon > 0$ and $\delta > 0$, there exists a $n_o > 0$ and a $t_o <$ such that for all t t $_o$ and n > n_o:

Pr[$| p_m(t) - 1 | < \varepsilon$] > 1 - δ

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

$$\mathbf{Pr}\{ \left| \frac{X_1 + X_2 + \ldots + X_t}{t} - \mu \right| > \varepsilon \} \varnothing 0 \text{ as } t \varnothing .$$

Let X_t be the indicator function such that:

X_t = 1 if α_i was rewarded the t^{th} time α_i was chosen,

= 0 if α_i was penalized the t^{th} time α_i was chosen,

and let Y_t^i be defined as in Theorem 4.1.

Hence by the weak law we have that for a given $\delta > 0$ there exists an $M_i <$, such that if α_i is chosen at least M_i times:

$$\mathbf{Pr}\{\, | d'_i(t) - d_i | < \frac{h}{2} \,\} > 1 - \delta.$$

Let $M = \text{Max}_{1\ i\ r}\ \{M_i\}$. Since h is strictly positive, it is clear that for all j   m and for all t, if $\text{Min}_{1\ i\ r}\ \{ Y_t^i \} > M$ then

$$\mathbf{Pr}\{\, |d'_m(t) - d_j| > {}^{h}/_{2}\} > 1 - \delta.$$

By Theorem 4.1 we know that we can define t_o and n_o such that for all i, all $t > t_o$ and all $n > n_o$

$$\mathbf{Pr}\ \{ Y_t^i > M \} \quad 1 - \ \delta. \qquad (4.9)$$

Thus if all actions are chosen at least M times, each of the d'_i will be in an ${}^{h}/_{2}$ neighbourhood of d_i with an arbitrarily large probability. But we know that :

$$d_m - {}^{h}/_{2} \quad > d_m - h \qquad \text{because } h > 0$$
$$d_i \qquad \text{for all i } \ m.$$

By the law of total probability , if B^c is the complement of the event B,

$$\mathbf{Pr}\ \{A\} = \mathbf{Pr}\ \{A \mid B\}\ \mathbf{Pr}\ \{B\} + \mathbf{Pr}\ \{A \mid B^c\}\ (1 - \ \mathbf{Pr}\ \{B\})$$

Since probability is a continuous set function we have:

$$\lim_{t\ \varnothing} \mathbf{Pr}\{A\} = \lim_{t\ \varnothing} \mathbf{Pr}\ \{A|B\} \lim_{t\ \varnothing} \mathbf{Pr}\ \{B\}$$
$$+ \lim_{t\ \varnothing} \mathbf{Pr}\ \{A \mid B^c\}\ (1- \lim_{t\ \varnothing} \mathbf{Pr}\ \{B\}).$$

But since each term is positive, the above can be simplified to

$$\lim_{t\ \varnothing} \mathbf{Pr}\{A\} \qquad \lim_{t\ \varnothing} \mathbf{Pr}\ \{A|B\} \lim_{t\ \varnothing} \mathbf{Pr}\ \{B\} \qquad (4.10)$$

Let A and B be the following events:

$A + | p_m - 1 | < \varepsilon$, and,

$$B + \mathrm{Max}_i \{ \left| d'_i(t) - d_i \right| \} < {}^{h}/_{2}.$$

Then clearly,

$$\mathbf{Pr}\{A \mid B\} = \mathbf{Pr}\{ \left| p_m - 1 \right| < \varepsilon \mid \mathrm{Max}_i \{ \left| d'_i(t) - d_i \right| \} < {}^{h}/_{2} \}.$$

Now using Theorem 4.1 and the monotone property, we know that

$$\lim_{t\, \varnothing} \ \mathbf{Pr}\{ A \mid B\} \ \varnothing\ 1$$

primarily because we can select a resolution parameter large enough to satisfy both the property and the Theorem 4.1. Furthermore, we know by Theorem 4.1 and (4.9) that $\lim_{t\, \varnothing}$ **Pr{**B**}**∅1-δ.

By (4.10) $\lim_{t\, \varnothing}$ **Pr** $\{ \left| p_m - 1 \right| < \varepsilon \}$ 1- δ for all $n > n_o$ and the theorem is proved.

With the proof complete, the next step is to use simulations of the automata in stochastic environments to illustrate some of the advantages of the discrete versions when compared to their continuous counterparts.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

V.1 Rate of Convergence and Accuracy

The problem with performing simulation results can be typified with the following: no matter how good an algorithm is, there could always be a single case in which another algorithm converges quicker and with a better accuracy. The approach taken to measure the performance of automata was to compare the DEAs to their continuous counterparts in environments which have been used as benchmarks by the inventors of their continuous counterparts Thathachar *et. al.* [11, 26, 28-31].

When considering these experimental results, the following property of learning automata should be kept in mind. If the objective is to pick the best action, an algorithm can generally be made very accurate if the rate of convergence is slowed down. Alternately, if accuracy is sacrificed, the rate of convergence can be quickened.

During the course of our study, simulations were performed to compare the rates of convergence of the discrete and continuous versions of both the Estimator Algorithms. To balance this trade off between speed and accuracy, the schemes were required to achieve a standard rate of accuracy of making no erroneous convergences in 100 experiments. The value of the internal parameter (λ for continuous, and n for discrete) was tuned to find the one which yielded the fastest convergence and which simultaneously satisfied the above stated standard rate of accuracy. Thus in the case of the continuous algorithms, the **largest** value of λ which yielded no errors during the test was used, and in the case of the discrete algorithms, the **smallest** value of the internal parameter, n, was the value reported. These parameters were then used to check the rates of convergence of the respective algorithms.

An algorithm was said to have converged whenever the probability of choosing an action exceeded 0.99. If the automaton converged to the best action (i.e., the one with the highest probability of being rewarded), it was said to have converged accurately. Table 3 summarizes the comparison of Discrete and Continuous Estimator algorithms in a 10 action environment. These environments were the same ones used to compare the continuous estimator algorithms to the L_{RI} scheme [28]. The estimator algorithms sampled all 10 actions, 10 times each, to initialize the estimate vector. These extra 100 iterations are also included in the results presented in each table.

The discrete algorithms always out-performed the corresponding continuous ones; typically the Continuous Pursuit algorithm is about 40 % slower than the discrete version. For the Continuous TSE Algorithm, the range is from 4 to 50 % slower that its discrete counterpart. For example, in the environment referred to as E_A, the DTSE Algorithm takes 207 iterations to reach the end state and the TSE Algorithm takes 310. When the 100 iteration used to initialize the automaton are not considered, the discrete version takes 107 iterations compared to 210. This is a decrease by a factor of approximately 2.

In the next set of simulations, the automata were placed in various two action environments similar to the one used to test the discrete version of the L_{RI} scheme [27]. The probability of reward for one action was fixed at 0.8 for all simulations. The probability of reward for the second action was increased from 0.2 to 0.775. Before starting the algorithm, estimates for **D**' were obtained by selecting each action 10 times, as was done by Thathachar and Sastry [31]. These extra 20 iterations were then included in the total number of iterations. The ensemble average results are shown in Table 4 and 5.

Table 3
Comparison of the Discrete and Continuous Estimator Algorithms
in Benchmark Ten-Action Environments

Environment	Algorithm	Continuous	Discrete
$\mathbf{E_A}$	Pursuit	1140	799
	TSE	310	207

$\mathbf{E_B}$	Pursuit	2570	1770
	TSE	583	563

Reward probabilities 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

When the difference in the probability of reward is marginal, both the discrete algorithms converge about twice as fast as their continuous counterparts. For example, with $d_1 = 0.8$ and $d_2 = 0.2$ the TSE Algorithm takes an average of 28.8 iterations and the DTSE Algorithm takes 24 iterations. However, twenty of these iterations are due to the initialization process. So after the estimates of reward are initialized, the TSE Algorithm takes 8.8 iterations to converge and the DTSE Algorithm takes 4 iterations to converge. For more difficult environments such as the one in which $d_1 = 0.8$ and $d_2 = 0.775$, the extra 20 iterations become negligible. In this case the TSE Algorithm takes 8,500 iterations for to converge, the DTSE Algorithm requires only 5,600. The advantage of discretizing is obvious.

V.2 CPU Factors

Note that apart from the computational gain observed in the mean number of iterations, the actual computing effort involved in the discrete scheme is significantly less because the probability updates at each iteration are not multiplicative. To illustrate this, the programs that were used for all algorithms were rendered identical except for the procedure that updates the probability vector. For each algorithm, the amount of CPU time used by the procedure that updated the probability vectors was monitored. The algorithms were given identical tasks, in the sense that they were all required to execute approximately the same number of iterations (within 4 %). The average amount of CPU time per iteration was calculated and the results are presented in Table 6.

Table 4
The Number of Iterations Until Convergence
in Two-Action Environments for the Pursuit Algorithms

Probability of Reward		Mean Iterations	
Action 1	Action 2	**Continuous**	**Discrete**
0.800	0.200	22	22

0.800	0.400	42	39
0.800	0.600	148	125
0.800	0.700	636	357
0.800	0.750	2980	1290
0.800	0.775	6190	3300

Table 5
The Number of Iterations Until Convergence
in Two-Action Environments for the TSE Algorithms

Probability of Reward		Mean Iterations	
Action 1	Action 2	**Continuous**	**Discrete**
0.800	0.200	28.8	24.0
0.800	0.400	37.	29.0
0.800	0.600	115.	76.
0.800	0.700	400.	380.
0.800	0.750	2200.	1200.
0.800	0.775	8500.	5600.

Because the rest of the program was identical for all the four schemes, the probability vectors used for the discrete algorithms had to be implemented as real numbers as opposed to integers. Thus the advantage observed in Table 6 reflects a decrease in the complexity of the discretized scheme. We believe that this can be attributed to such things as the reduction in the amount of floating point multiplications that are done. Thus, **even without using the integer representation**, discrete algorithms take from 50 to 75 percent of the time that their continuous counterparts take.

Table 6
CPU Time Used per Iteration in a Ten-Action Environment

Algorithm	**Continuous**	**Discrete**

Pursuit	2.9 msec	2.3 msec
TSE	14 msec	7.2 msec

V.3 Adaptive Features

One final set of simulations was run to illustrate a desirable property that the TSE Algorithm has, which the Pursuit algorithms do not have. If the internal parameter remains the same, but the task is made progressively harder, the estimator algorithms adapt by increasing their number of iterations more than the Pursuit algorithms.

In Tables 7 and 8, the difference in reward probabilities is decreased by a factor of four between the subsequent environments reported in the tables. Both TSE algorithms respond by increasing their number of iterations by roughly the same factor, namely, four. The number of iterations increases in a multiplicative fashion, in the sense that this number increases by a constant factor as the difference between the penalty probabilities is decreased. The Pursuit algorithms respond in a less favourable way. In this case it seems as if the number of iterations increases in an additive fashion. In the continuous case, approximately 100 more iterations are required as the task gets progressively harder. Interestingly enough, discretizing the probability space retains this property inasmuch as the DPA responds in an analogous way.

Table 7

Convergence and Error Rate for a Fixed Parameter for the TSE Algorithms

Probability of Reward		Continuous		Discrete	
Action 1	Action 2	Iterations	Errors	Iterations	Errors
0.800	0.400	78	0 %	56	0
0.800	0.700	377	0 %	364	0
0.800	0.775	1190	20 %	1010	17 %

Table 8

Convergence and Error Rate for a Fixed Parameter for the Pursuit Algorithms

Probability of Reward		Continuous		Discrete	
Action 1	Action 2	Iterations	Errors	Iterations	Errors
0.800	0.400	526	0 %	383	0 %
0.800	0.700	616	0 %	437	0 %
0.800	0.775	765	24 %	625	25 %

VI. CONCLUSION

As software is becoming increasingly complex there is a desire for general methods that solve a wide variety of problems. Consider the example of a computer controlled switching circuit. One routine may decide how to handle the routing, a second may optimize the call processing queues, and a third may deal with the storage of customer files. Rather that have three separate approaches to be designed, implemented, debugged, and documented, it would be much easier if a single algorithm could optimize all of these situations. VSSA can handle all of these problems. Automata are adaptable, and as such, represent a fault tolerant approach.

The rate at which VSSA converge has been a limiting factor in their implementation in the past. Discretizing provides a general method of improving their performance.

Estimator Algorithms are among the quickest stochastic learning automata known to date. In this paper we have considered discretizing them and shown that ε-optimality is preserved when they are discretized. In fact, in some environments the discrete versions requires only about 50% of the number of iterations required for its continuous counterpart. As well the amount of CPU time used per iteration can be reduce by a factor ranging between 50 % and 75 %.

REFERENCES

[1] Baba, S., Soeda, S.T., and Sawaragi, Y., "An Application of Stochastic Automata to the Investiment Game," *Int. J. Syst. Sci.*, vol. 11, no. 12, pp 1447-1457, Dec. 1980.

[2] Karlin, S., Taylor, H. M., *A First Course on Stochastic Processes* , 2d ed.,Academic Press , 1974.

[3] Lakshmivarahan, S., *Learning Algorithms Theory and Applications,* Springer-Verlag, 1981.

[4] Lakshmivarahan, S., "Two Person Decentralized Team With Incomplete Information", *Applied Mathematics and Computation*, Vol. 8, pp.51-78, 1981.

[5] Lakshmivarahan, S., and Thathachar, M.A.L., "Absolutely Expedient Algorithms for Stochastic Automata", *IEEE Trans. on Syst. Man and Cybern.*, Vol. SMC-3, 1973, pp. 281-286.

[6] Lanctôt, J. K.,*Discrete Estimator Algorithms: A Mathematical Model of Computer Learning*, M.Sc. Thesis, Department of Mathematics and Statistics, Carleton University, Ottawa, Canada, 1989.

[7] Meybodi, M.R.,*Learning Automata and Its Application to Priority Assignment in a Queueing System With Unknown Characteristic,* Ph.D. Thesis, School of Electrical Engineering and Computing Sciences, University of Oklahoma, Norman, Oklahoma.

[8] Meybodi, M.R.,Lakshmivarahan, S., "A Learning Approach to Priority Assignment in a Two Class M/M/1 Queuing System with Unknown Parameters", *Proc. Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, 1983.

[9] Motorola, *MC68000 16/32-Bit Microprocessor Programmer's Reference Manual,* 4th ed., Prentice-Hall, 1984.

[10] Motorola, *MC68020 32-Bit Microprocessor*, 2d ed., Prentice-Hall, 1985.

[11] Mukhopadhyag, S., Thathachar, M.A.L., "Associative Learning of Boolean Functions", *IEEE Systems Man and Cybernetics Transaction*, Sept/Oct 89, pp.1008-1015.

[12] Narendra, K.S., and Thathachar, M.A.L., *Learning Automata*, Prentice-Hall, 1989.

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

[14] Narendra, K.S., and Thathachar, M.A.L., "On the Behaviour of a Learning Automaton in a Changing Environment With Routing Applications", *IEEE Trans. on Syst. Man and Cybern.*, Vol. SMC-10, 1980, pp.262- 269.

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

[16] Narendra, K.S., and Lakshmivarahan, S, "Learning Automata : A Critique", *Journal of Cybernetics and Information Sciences*, Vol. 1, 1987, pp.53-66.

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

[18] Oommen, B.J., and Christensen, J.P.R., "Epsilon-Optimal Discretized Linear Reward-Penalty Learning Automata", *IEEE Transactions on Systems, Man and Cybernetics,* Vol. SMC-18, May/June 1988, pp. 451-458.

[19] Oommen, B.J., and Thathachar, M.A.L., "Multiaction Learning Automata Possessing Ergodicity of the Mean", *Information Sciences,* Vol. 35, No. 3, June 1985, pp. 183-198.

[20] Oommen, B.J., and Lanctôt, J.K., "Discretized Pursuit Linear Reward-Inaction Learning Automata", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-20, July/August 1990, pp. 431-438.

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

[22] Oommen, B. J., and Ma, D. C. Y., "Deterministic Learning Automata Solutions to the Equi-Partitioning Problem", *IEEE Transactions on Computers*, Vol. 37, January 1988, pp.2-14.

[23] Oommen, B.J., and Ma, D.C.Y., "Fast Object Partitioning Using Stochastic Learning Automata". *Proceedings of the 1987 International Conference on Research and Development in Information Retrieval*, New Orleans, June 1987.

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

[25] Ross, Sheldon M., *Introduction to Probability Models*. 3 rd Ed. San Diego: Academic Press, 1985.

[26] Sastry, P.S., *Systems of Learning Automata: Estimator Algorithms Applications*, Ph.D. Thesis, Dept of Electrical Engineering, Indian Institute of Science, Bangalore, India, June 1985.

[27] Thathachar, M.A.L., and Oommen, B.J., *Discretized Reward-Inaction Learning Automata*, Journal of Cybernetics and Information Sciences, Spring 1979, pp. 24-29.

[28] Thathachar, M.A.L., and Sastry, P.S., "A New Approach to Designing Reinforcement Schemes for Learning Automata", presented at *IEEE Int. Conf. on Cybernetics and Society*, Bombay, India, January 1984.

[29] Thathachar, M.A.L., and Sastry, P.S., "Estimator Algorithms for Learning Automata", *Proc Platinum Jubilee Conference on Systems and Signal Processing*, Dept of Electrical Engineering, Indian Institute of Science, Bangalore, India, December 1986.

[30] Thathachar, M.A.L., and Sastry, P.S., "A Class of Rapidly Converging Algorithms for Learning Automata", *IEEE Trans. Syst. Man and Cybern.*, Vol SMC-15, pp. 168-175, January 1985.

[31] Thathachar, M.A.L., and Sastry, P.S., "Pursuit Algorithm for Learning Automata", Unpublished document. Copy is availiable from the second author of this paper.

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

[33] Tsetlin, M.L., *Automaton Theory and the Modelling of Biological Systems*, New York and London, Academic, 1973.