National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-59045-9

1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

DISCRETE ESTIMATOR ALGORITHMS:
A MATHEMATICAL MODEL OF MACHINE LEARNING
by
J. KEVIN LANCTOT

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Science
Department of Mathematics

Carleton University
Ottawa, Ontario
December 6, 1989

© copyright
1989, J. Kevin Lanctôt

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-59045-9

To my father,
Edmond A. Lanctôt

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis
DISCRETE ESTIMATOR ALGORITHMS:
A MATHEMATICAL MODEL OF MACHINE LEARNING
submitted by
J. KEVIN LANCTOT
in partial fulfilment of the requirements
for the degree of Master of Science

Chair, Department of Mathematics

Thesis Supervisor

External Examiner

Carleton University
December 20, 1989

ABSTRACT

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 algorithms known. They are characterized by the use of running estimates of the probabilities of being rewarded. This paper investigates the improvements gained by rendering the various estimator algorithm 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. Furthermore, experimental results indicate that the discrete modifications improve the performance of these algorithms.

ACKNOWLEDGEMENTS

I would like to thank the following people who at some time or other read and commented on parts of this thesis: Jean Marc Auprix, Dwight Deugo, Robert Harland, M. A. L. Thathachar, and Radha Valiveti. I am especially grateful to Professor John Dixon for his work noting an oversight in the proof of Theorem 4.1.

A special thank-you to Professor John Oommen, whose relentless enthusiasm and constant help made this thesis possible.

TABLE OF CONTENTS

Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	viii
List of Illustrations	x
List of Symbols	xi
Chapter 1 INTRODUCTION TO LEARNING AUTOMATA	
1.1 Artificial Intelligence and Learning Automata	1
1.2 Definition of Deterministic Automata	2
1.3 Example of an Automaton in a Learning Situation	5
1.4 Generalization to Stochastic Automata	7
1.5 Simplifications of Stochastic Automata	10
1.6 Learning Automata and Stochastic Environments	13
1.7 Definitions of Learning Criteria	16
1.8 Varieties of Learning Automata	19
1.9 Variable Structure Stochastic Automata	23
1.10 Applications of LA	27
1.10.1 Telephone Traffic Routing	27

1.10.2 Priority Assignments in a Queue	29
1.10.3 Object Partitioning	30
Chapter 2 DISCRETE LEARNING ALGORITHMS	
2.1 Introduction	35
2.2 Existing Discrete Automata	36
2.3 Motivation for Discretization	44
Chapter 3 ESTIMATOR ALGORITHMS	
3.1 Characterization and Motivation	50
3.2 The Pursuit Algorithm	51
3.3 The TS Estimator Algorithm	56
Chapter 4 DISCRETE ESTIMATOR ALGORITHMS	
4.1 Introduction	62
4.2 The Discrete Pursuit Algorithm	63
4.3 The Discrete TS Estimator Algorithm	68
4.4 Proof Of Convergence	73
Chapter 5 SIMULATION RESULTS	
5.1 Rate of Convergence and Accuracy	81
5.2 CPU Factors	85
5.3 Adaptive Features	86

Chapter 6 CONCLUSION

6.1 Present Role of Discrete Estimator Algorithms	89
6.2 Future Prospects of Discrete Estimator Algorithms	90

REFERENCES	91
------------	----

LIST OF TABLES

TABLE	DESCRIPTION	PAGE
1	Functions for the $L_2, 2$ Automaton	6
2	Probability Distribution of Next State and Output Given that the Present State is q_1 and that the Automaton is Rewarded	12
3	A Three Action Environment	18
4	Varieties of Updating Schemes	25
5	A Comparison of Updating Rules for Various Two Action Learning Automata	26
6	Simulation of Object Migration	33
7	The DL_{RI} Compared with the Tsetlin and Krinsky Automata	38
8	A Comparison of Updating Rules For Various Types Of Learning Automata	40
9	A Comparison of Asymptotic Properties of Linear VSSA	41
10	Comparison of Updating Rule for L_{RI} and DL_{RI}	42
11	Comparison of DL_{RI} and DL_{IP} Automata	43

12	Comparison of the Basic Features of Estimator and Non-estimator Algorithms	52
13	Performance of the Pursuit Algorithm and the LRI	56
14	Performance of the TS Estimator Algorithm and LRI	61
15	Comparison of Discrete and Continuous Estimator Algorithms 10 Action Environments	83
16	The Number of Iterations Until Convergence in Two Action Environments for the Pursuit Algorithm	84
17	The Number of Iterations Until Convergence in Two Action Environments for the TS Estimator Algorithm	84
18	CPU Time Used per Iteration in a Ten Action Environment	86
19	The Number of Iterations and Error Rate for a Fixed Parameter for the TS Estimator Algorithms	87
20	The Number of Iterations and Error Rate for a Fixed Parameter for the Pursuit Algorithms	87

LIST OF ILLUSTRATIONS

FIGURE	DESCRIPTION	PAGE
1.	Representation of an Automaton	4
2	State Transition and Output Digraph Representation of the $L_{2,2}$ Automaton	7
3	State Transition and Output Matrices for the $L_{2,2}$ Automaton	8
4	A Digraph Representation of a Stochastic Automaton.	9
5	A Matrix Representation of a Stochastic Automaton	11
6	Conversion to Deterministic Output Function	12
7	A Learning Automaton Interacting with the Environment	14
8	Classification of Existing Learning Automata	20
9	State Transitions for the Tsetlin's Automaton	21
10	State Transitions for the Krylov Automaton	23
11	Example of a Network for Traffic Flow	28

12	Jobs Size and Mean Processing Time Automaton	30
13	A Comparison of the Probability of Choosing the Optimal Action for a Continuous and a Discretized Learning Algorithm	45

LIST OF SYMBOLS

LATIN	DESCRIPTION
A	the set of all possible outputs or actions of an automaton
ADLIP	Absorbing Discrete Linear Inaction-Penalty scheme (see Table 8)
ADLRP	Absorbing Discrete Linear Reward--Penalty scheme (see Table 8)
B	the set of all possible inputs to an automaton $B = \{0,1\}$ where 0 is a reward and 1 is a penalty
$C(t,m)$	the numbers of combinations when m items are chosen among t elements
c_i	the probability of the automaton being penalized given it selected action α_i
D	the set that characterizes the probability of reward for each action
d_b	the unique maximum for the set D
d_i	the probability of the automaton receiving a reward given that action α_i was just selected
$d_i(t)$	the probability of the automaton receiving a reward at time t given that action α_i was selected
DEA	Discrete Estimator algorithm

DLIP	Discrete Linear Inaction-Penalty scheme (see Table 8)
DLRI	Discrete Linear Reward-Inaction scheme (see Table 8)
DLRP	Discrete Linear Reward--Penalty scheme (see Table 8)
DPA	Discrete Pursuit Automaton
DSTEAs	Discrete TS Estimator algorithm
$D'(t)$	the estimate vector (of the probability of each action being rewarded)
e_m	the unit vector whose m^{th} component is unity
F	the function of an automaton that determines the next internal state
f	monotonic, increasing function, satisfying $f(0) = 0$ and $f(1) = 1$
FSSA	fixed structure stochastic automaton
G	the function of an automaton that determines the output
h	the difference the probability of reward of the optimal and next to optimal action
IP	an Inaction-Penalty scheme (see Table 4)
k_i	an arbitrary integer
LA	Learning Automaton
$L_{2,2}$	a simple learning automaton see page 5
LIP	Linear Inaction-Penalty scheme (see Table 5)
LRI	Linear Reward-Inaction scheme (see Table 5)

LRP	Linear Reward--Penalty scheme (see Table 5)
M	an arbitrary constant, each action is chosen at least this many times
M ₀	the probability of being penalized if all actions are chosen with equal probability
M(t)	the probability of an automaton being penalized at time t
m	the component of the estimate vector that has the largest magnitude except Theorem 4.1 where it is an arbitrary term of the summation
N	the index for the states in a fixed structure automaton
n	the resolution parameter for discrete VSSA
n ₀	an arbitrary integer
o _i	an object in the object migration automaton
P(t)	the probability distribution over the set of actions
p _i (t)	the probability that the automaton will pick action α_i at time t
p_m^∞	the value to which $p_m(t)$ converges to
$\Re$	the set of real numbers
RI	a Reward-Inaction scheme (see Table 4)
RP	a Reward-Penalty scheme (see Table 4)
r	the number of actions

Q	the set of all possible internal states of an automaton
$q(t)$	internal state of the automaton at time
q_i	the i^{th} internal state of the automaton, an element of Q
$S_{ij}(t)$	is an indicator function, $S_{ij}(t) = 1$ if $d'_i(t) > d'_j(t)$ $= 0$ if $d'_i(t) \leq d'_j(t)$.
T	the learning algorithm
t	time, or the number of times the automaton has received an input
t_0	a point in time
TSEA	TS Estimator algorithm (page 56)
VSSA	variable structure stochastic automata
$W_i(t)$	the number of times the i^{th} action has been rewarded up to time t .
X_t^i	the indicator function which has the value 1 if the i^{th} action is rewarded the t^{th} time α_i was chosen, and 0 otherwise.
Y_t^i	the number of times the i^{th} action was chosen up to time t .
$Z_i(t)$	the number of times the i^{th} action has been chosen up to time t .
Z	the integers

GREEK	DESCRIPTION
$\alpha(t)$	action the automaton takes at time t
α	an element of A , the set of actions
α_i	the i^{th} action, an element of A
β	an element of B , the set of input to the automaton
$\beta(t)$	input the automaton receives at time t
Δ	the smallest change in probability, a discrete algorithm can take
δ	an arbitrary small positive real number
ϵ	an arbitrary small positive real number
ϕ_{ij}	the transition of an automaton which corresponds to: go to state q_i and pick action α_j
λ	the internal parameter of a continuous VSSA
μ	mean
ψ	replaces $1 - 1/2 p_1(0)$ in Theorem 4.1
σ^2	variance
θ	the largest integer multiple of Δ that any one component of the probability vector can decrease in one iteration
χ	a positive integer

CHAPTER 1

INTRODUCTION TO LEARNING AUTOMATA

1.1 Artificial Intelligence and Learning Automata

A primary goal of artificial intelligence is to extend the use of computers to solve problems typically handled by the human intellect. Rather than requiring a complete database to solve a particular class of problems, ideally the intention is to arrive at the best solution with the information available. Tsetlin created an archetype for a possible solution to this problem; he developed a model of computer learning which is now called the learning automaton (LA). Using this, Tsetlin modelled a biological type of learning [45]. He abstracted the situation where an organism is born with relatively little initial knowledge and learns which actions are appropriate through trial and error. While interacting with the 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 then uses this information to decide which action to take next, and the cycle continues.

In the study of LA, the environment is typically modelled as one in which each action has a probability of being rewarded, rather than always being rewarded or penalized. The LA attempts to find which strategy is the optimum, that is, which action has the maximum probability of being rewarded. The key point is that the probability of success for each action is unknown to the automaton; it adapts itself to the environment by learning the optimal action.

Chapter 1 of this thesis will give an introduction to LA. The aim is to motivate the definition of variable structure stochastic automata. This is the particular class of automata that will be dealt with in the rest of this thesis. An indication of their variety and applications will also be given. Typically, the performance of these algorithms is compared to automata developed by Tsetlin, and Krylov, so these automata will be introduced along the way. Chapter 2 will describe the family of discrete LA which will be the focus of all the results presented in this thesis. This is a general method of improving some properties of variable structure stochastic automata such as their rates of convergence. Chapter 3 will present a recently introduced class of rapidly converging learning automata called estimator algorithms, which are among the fastest converging algorithms known. Chapter 4 will introduce a new class of LA called discrete estimator algorithms. These algorithms have the advantages of both the previous approaches. A proof of their convergence will be given. Finally, Chapter 5 will present some simulation data that compares implementations of both the continuous and discrete versions of the estimator algorithm schemes and demonstrates the superiority of the latter schemes.

1.2 Definition of Deterministic Automata

In this thesis, the basic approach used to study learning organisms can be called systems theory. Indeed, to quote Booth, "Systems theory is based on the assumption that the external behaviour of any physical device can be described by a suitable mathematical model. The resulting mathematical model is called a system." [2]. In this context, the particular system studied is a learning system. In order to analyse learning, the milieu in

which the learning is to take place must be defined. Measuring over time seems inappropriate; it does not take into account the number of chances for the system to learn within a given time interval. A situation where learning can take place must be abstracted with the concepts of input and output being well defined. An input can be thought of as a chance to learn and the output can be thought of as the response the system gives to the input.

Even the most naïve definition of learning accepts that with the same input, different outputs may occur before and after learning has taken place. So in addition to input and output, our prototype of machine learning must have a component that can change to reflect that learning has taken place [35]. This entity will be called the internal state. One of the simplest mathematical constructs that includes inputs, outputs and internal states is the deterministic automaton.

'Automaton' means literally 'something that directs itself' [35]. Among other devices, the digital computer satisfies the constraints of the deterministic automaton. With this in mind, the following is considered. The automaton consists of three sets: the set of all possible inputs, B ; the set of all possible internal states or 'memory', Q ; and the set of all possible outputs, A . The automaton processes the inputs it receives sequentially. When presented with an input, the automaton produces an output based on its present internal state. Then the automaton adjusts its internal state based deterministically on the input and its present state. In other words there is a function F that maps the present state and the input into the next state, and a function G that maps the present state into the output. Using the above, the automaton can be described at any precise moment of time. The t^{th} input is denoted by $\beta(t)$. The internal state at this instant by $q(t)$ and the output by

$\alpha(t)$.¹ A representation of an automaton is presented in Figure 1. With these three sets and two functions defined, the description of the finite deterministic automaton is complete. Using these terms the automaton can be described mathematically.

Figure 1. Representation of an Automaton

Definition 1.1: A deterministic automaton² is a set $\{B, Q, A, F, G\}$ where B , Q , and A are sets; F and G are deterministic functions, such that $F: B \times Q \rightarrow Q$ and $G: Q \rightarrow A$. Thus at any instant t , given $\beta(t) \in B$, $q(t) \in Q$, and $\alpha(t) \in A$, the automaton is constrained by the relationship: $q(t+1) = F(\beta(t), q(t))$, and $\alpha(t) = G(q(t))$.

Definition 1.2 : A finite deterministic automaton is a deterministic automaton consisting of the set $\{B, Q, A, F, G\}$ such that B , Q , and A are all finite sets.

¹Strictly speaking t does not represent time, as is generally suggested. Here t means the t^{th} input. This is to distinguish the fact that as time passes from say 12:00.00 to 12:00.01 there may or may not have been an input; however, there is exactly one input, one output and one change of state in the transition from t to $t+1$.

²The definition that follows is a Moore automaton; there are other types such as the Mealy automaton, which is identical except that the function G is defined as $\alpha(t) = G(\beta(t), q(t))$. Since G is a function of the input as well, this means that the Mealy automaton needs an input before giving an output, whereas the Moore automaton can give its first output without an input. This subtle point becomes important for the working of LA for it must offer an initial guess to get feedback from the environment. The two can be made equivalent except for the timing of their initial outputs [2].

Throughout this thesis the cardinalities of B and A will be two, and r respectively. The cardinality of Q will vary, but it will be always be a function of n .

It can be seen that a digital computer satisfies the constraints of a finite automaton. The input is the data the computer is given. The internal states are all the possible contents in memory. The program is the function that governs the change of states and the output. In some ways, the processing that humans do is similar to that done by an automaton. The input is what our senses perceive, the output is our actions, and the change in internal state reflects our brain processing the information. What is hard to accept is that the changes are deterministic. This limitation makes it necessary to generalize deterministic automata to stochastic automata in order to more easily reflect a biological model of learning.

1.3 Example of an Automaton in a Learning Situation

For an illustration of the above consider the following simple problem. An automaton is given the task that it must choose between two actions. The result of taking one action is to be considered desirable (which from now on will be termed rewarded) and the result of the other action is to be considered undesirable (which will be called penalized). It is the task of the automaton to learn which of the two actions is the desirable one. One possible strategy is obvious. Try one action, if it is rewarded then pick it again next time; if it is penalized then try the other action for the next choice. An automaton model for this strategy will now be given.

The set of internal states will consist of $Q = \{q_1, q_2\}$. Intuitively, q_1 will represent "pick the first action", and q_2 will represent "pick the second

action". The set of outputs is $A = \{\alpha_1, \alpha_2\}$, where α_1 is action 1 and α_2 is action 2. Let $B = \{0,1\}$ where 1 corresponds to a penalty and 0 corresponds to a reward.³ Finally, the functions F and G must be defined. This is done in Table 1. This automaton is called the $L_{2,2}$ Automaton.

While the list of values for the next state and output functions completely characterize the automaton, there are two other traditional ways of presenting the information: (1) digraph⁴ representation and (2) matrix representation.

Table 1.
Functions for the $L_{2,2}$ Automaton

<u>Next Internal State Function</u>		<u>Output Function</u>
$F(0, q_1) = q_1$	$F(0, q_2) = q_2$	$G(q_1) = \alpha_1$
$F(1, q_1) = q_2$	$F(1, q_2) = q_1$	$G(q_2) = \alpha_2$

With the digraph representation, the various states are represented by nodes. The transitions that are possible are represented by arcs. The state transition function is represented by a digraph for each possible input, while the output function is represented by a single digraph. The automaton characterized in Table 1 is represented in digraph form in Figure 2.

The other method of representing the automaton is with matrices. There is one matrix for each digraph. The automaton characterized in Table 1 is represented in matrix form in Figure 3.

³Although this notation is the most prevalent in the literature, please note that it is not universal; in some papers, particularly those dealing with estimator algorithms, 0 corresponds to a penalty and 1 corresponds to a reward.

⁴A digraph is a graph with directed edges. See [3] for a more rigorous definition.

State Transitions Digraphs:

Output Function Digraph:

Figure 2 State Transition and Output Digraph Representation of the $L_{2,2}$ Automaton

1.4 Generalization to Stochastic Automata

Initially, it is hard to understand what sort of computational ability or limitations automata have. It has been shown [35] that a finite deterministic automaton is equivalent to a statement in predicate calculus. Recall that predicate calculus is built up of sentences of finite length with the connectives: "and", "or", "if", "then", "not", and "if and only if".⁵ Intuitively this equivalence makes sense. The automaton is constructed as if it is saying "If (given this input **and** in this state) **then** (select this output **and** go to this next state)." For example, the automaton described in Table 1 represents the clause : "If $\{ \beta(t) = 0 \text{ and } q(t) = q_1 \}$ **then** $\{ q(t+1) = q_1 \text{ and } \alpha(t) = \alpha_1 \}$ **and if** $\{ \beta(t) = 0 \text{ and } q(t) = q_2 \}$ **then** $\{ q(t+1) = q_2 \text{ and } \alpha(t) = \alpha_2 \}$

⁵ For more rigorous definition see [5]

and if { $\beta(t) = 1$ and $q(t) = q_1$ } then { $q(t+1) = q_2$ and $\alpha(t) = \alpha_2$ }

and if { $\beta(t) = 1$ and $q(t) = q_2$ } then { $q(t+1) = q_1$ and $\alpha(t) = \alpha_1$ }

State Transition Matrix

$$F_{\beta=0} = \begin{matrix} & \begin{matrix} q_1 & q_2 \end{matrix} \\ \begin{matrix} q_1 \\ q_2 \end{matrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix} \quad F_{\beta=1} = \begin{matrix} & \begin{matrix} q_1 & q_2 \end{matrix} \\ \begin{matrix} q_1 \\ q_2 \end{matrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{matrix}$$

Output Matrix

$$G = \begin{matrix} & \begin{matrix} \alpha_1 & \alpha_2 \end{matrix} \\ \begin{matrix} q_1 \\ q_2 \end{matrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix}$$

Figure 3. State Transition and Output Matrices for the $L_{2,2}$ Automaton

This indicates that the operation of the automaton can be described in terms of predicate calculus. Furthermore, its behaviour is completely predictable; in other words, given two identical sequences of inputs the automaton will always give the same two sequences of outputs. This is the sort of rigid learning that typifies digital computers. This inflexibility will be relaxed by introducing stochastic automata.

Stochastic automata are a generalization of deterministic automata. For a deterministic automaton F and G are deterministic functions. In this case, if the internal state and the input are known, the output and the next state can be predicted with certainty. For stochastic automata this condition

is relaxed. Given an internal state and an input, the output and next state may be random with a specified probability.

Although this generalization may seem to add a lot of complexity to automata, as seen in Figure 4, the digraphs lend themselves quite naturally to stochastic automata.

State Transitions Digraphs:

if rewarded

if penalized

Output Function Digraph:

Figure 4. A Digraph Representation of a Stochastic Automaton

Like the deterministic case, the arcs represent the possible state transitions. Arcs are weighted with the probability of the corresponding transition occurring. In order that these weights represent probabilities, the sum of the weights for arcs leaving a node must be unity. For example, in Figure 4, if the automaton is rewarded and it is in state q_1 , the automaton will stay there

with a probability of 0.6 and leave with a probability of 0.4. The digraph, for the case when the automaton is penalized, illustrates that the deterministic digraph is just a special case of a stochastic digraph where all the arcs are weighted with unity.

Transition matrices generalize to the stochastic case, as well. Instead of allowing only entries zero and one, entries are allowed to be any element of the interval $[0,1]$ with the constraint that the sum of entries in any row must equal one. The matrices in Figure 5 correspond to the digraphs in Figure 4.

For the matrix that represents the function F the entry in the i^{th} row and j^{th} column represents:

$$\Pr\{ q(t+1) = q_j \mid q(t) = q_i, \beta(t) = \beta \} \text{ where } \beta \in \{ 0, 1 \} \quad (1.1)$$

As well, for the matrix that represents the function G the entry in the i^{th} row and j^{th} column represents:

$$\Pr\{ \alpha(t) = \alpha_j \mid q(t) = q_i \}. \quad (1.2)$$

1.5 Simplifications of Stochastic Automata

Now although both F and G are allowed to be stochastic, the purpose of this section is to show that nothing is gained by allowing G to be stochastic; every automaton with both F and G stochastic is equivalent to an automaton with its state transition matrix stochastic and its output matrix deterministic. Consider the example presented in Figures 4 and 5. Table 2 illustrates the probability distribution of the transition and output states given that the automaton is rewarded and that it is in state q_1 .

The approach is to increase the number of internal states in the new automaton in order to reflect every possible combination of internal states

State Transitions Matrices

if rewarded

$$F_{\beta=0} = \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$$

if penalized

$$F_{\beta=1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Output matrix

$$G = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

Figure 5. A Matrix Representation of a Stochastic Automaton

and output in the old automaton (where G is stochastic). The new state ϕ_{ij} will represent the automaton being in the old state q_i and stochastically selecting the output α_j . The new output function stipulates that state ϕ_{ij} deterministically picks action α_j . When this is done for all the possible state transitions and output choices made at q_1 , the automaton presented in Figure 4 is equivalent to the one presented in Table 2 and Figure 6.

All that remains is to show that every possible pair of internal state and output transitions is the same whether it is calculated in the new automaton with the deterministic output function or in the original one with stochastic output function. The new state space will be the cartesian product of the set of states of the original automaton and the set of actions, and will be of the form ϕ_{ij} . The entries of the new G matrix would be of the form: $\Pr\{ \alpha(t) = \alpha_k \mid q(t) = \phi_{ij} \}$ $= 0$ when $k \neq j$
 $= 1$ when $k = j$.

Table 2--Probability Distribution of Next State and Output

Given that the Present State is q_1 and that the Automaton is Rewarded		
<u>next state</u>	<u>output</u>	<u>probability</u>
q_1	α_1	$0.6 \times 0.9 = 0.54$
q_1	α_2	$0.6 \times 0.1 = 0.06$
q_2	α_1	$0.4 \times 0.2 = 0.08$
q_2	α_2	$0.4 \times 0.8 = 0.32$

State Transitions Graph:

Output Function Graph:

Figure 6 Conversion to Deterministic Output Function

Hence the output of the new automaton is deterministic. Now consider an arbitrary transition from ϕ_{ij} to ϕ_{kl} , on receiving $\beta(t) = \beta$. The probability that this will happen is expressed as:

$$\Pr\{ \alpha(t+1) = \alpha_l \text{ and } q(t+1) = q_k \mid q(t) = q_i, \alpha(t) = \alpha_j, \beta(t) = \beta \} \quad (1.3)$$

Note that this probability is equivalent to the product :

$$\Pr\{ \alpha(t+1) = \alpha_l \mid q(t+1) = q_k, q(t) = q_i, \alpha(t) = \alpha_j, \beta(t) = \beta \} \cdot \Pr\{ q(t+1) = q_k \mid q(t) = q_i, \alpha(t) = \alpha_j, \beta(t) = \beta \} \quad (1.4)$$

Now for the original automaton, the output and next state depend only on the input and the present state, so after eliminating the extraneous terms from the above statement, (1.4) simplifies to the following:

$$\Pr\{ \alpha(t+1) = \alpha_j \mid q(t+1) = q_k \} \bullet \\ \Pr\{ q(t+1) = q_k \mid q(t) = q_i, \beta(t) = \beta \} \quad (1.5)$$

This is true because in the original automaton $\alpha(t+1)$ depends only on $q(t+1)$ and $q(t+1)$ depends only on $q(t)$ and $\beta(t)$. The first factor in equation (1.5) is in the form of equation (1.2), and is an element of G the output function. The second factor is in the form of equation (1.1) and is an element of F , the probability of the state transition function. Using this fact, equation (1.5) is equivalent to the probability of the event that the automaton goes from state q_i to state q_k on input β and then chooses the action α_j . So the new automaton is essentially equivalent to the old one.

With this simplification in mind, for the rest of this thesis, stochastic automata will be presented with their output functions deterministic.

1.6 Learning Automata and Stochastic Environments

One of the basic methods of mathematics is to try to decide which properties of a system are essential and to use abstractions of these properties to justify any statements made about the system. Imagine that the task was to decide the safest route to walk to work. A safe route would avoid streets with heavy traffic. To keep track of every car in the city would be a monumental task. A reasonable simplification would be to estimate the

probability of a car coming while crossing the street at the various points along the proposed routes. The safest path could then be based on crossing streets with a low probability of being used by cars. In doing this, the problem has been simplified by creating a stochastic model of an aspect of the environment. This is the route that designers of learning automata have taken: a learning automaton learns in the context of it interacting with a stochastic environment.

So, in order to understand what a learning automaton is, the next thing to be defined is how the stochastic environment is to be modelled. Rather than considering the environment as a pervasive entity, the input and output sets of the automaton form an interface with the environment. The output of the automaton is identical to the input of the environment and the input of automaton is exactly the output emitted by the environment. This is illustrated in Figure 7.

Figure 7: A Learning Automaton Interacting with the Environment

Definition 1.3 A stochastic environment will be defined as a 3-tuple of sets, $\langle A, B, D \rangle$. The sets A and B are identical to those of the

automaton. D is the probability of reward $D = \{d_1, d_2, \dots, d_r\}$, where $d_i(t) = \Pr\{\beta(t) = 0 \mid \alpha(t) = \alpha_i\}$.

For this thesis $B = \{0, 1\}$ where zero represents a reward and one represents a penalty. This is called a P-environment in literature pertaining to learning systems. Other models permit B to be a finite set, $\{0, a_1, a_2, \dots, a_{n-1}, 1\}$; this is called the Q-environment. Finally B can be the interval $[0, 1]$ in which case the model is an S-environment.

The set D characterizes the environment in the sense that its elements represent the probability that an element of A will be rewarded. The elements 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 varies with time, the environment is called non-stationary. If the set D is constant over time, then the environment is called stationary.

This thesis 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 cardinality of D is the same as that of A because there is a unique probability of being rewarded for each action in A and so D equals $\{d_1, d_2, \dots, d_r\}$. In order to give the learning automaton a well defined task, it is also assumed that there is a unique maximum for the set D called d_b . Mathematically, $d_b = \text{Max}_{1 \leq i \leq r} \{d_i\}$. This action 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. It bases its decision on the information gained by selecting actions, and getting feedback from the environment concerning whether the action was rewarded or penalized. This

is done in a sequential manner. The automaton picks an initial action. The environment rewards or penalizes it. The automaton then updates its internal state based on the feedback from the environment, and then selects another action. This cycle continues until the learning process is terminated.

How the automaton picks the next internal state and the next output is, in essence, the art and the science of designing learning automata. Rather than consider the two functions F and G that characterize the change of state and output as separate, they are often referred to together as T , the learning algorithm. Hence a learning automaton is defined equivalently as a 5-tuple $\{ A, B, Q, F, G \}$ or the 4-tuple $\{ A, B, Q, T \}$.

Definition 1.4 A learning automaton is defined as an automaton interacting with a stochastic environment.

1.7 Definitions of learning criteria

Now that the interaction between the automaton and the environment is understood, the next step is to set criteria for performance, to distinguish a good design from a bad one. Intuitively, the goal is for the automaton to arrive at the right answer as a result of the interaction. In the case of learning automata, this task is to find the best action, the one that has the highest probability of being rewarded. The probability that the automaton will pick the best action is denoted p_b . The first two definitions that follow are performance definitions. They characterize the automaton with respect to accuracy.

Definition 1.5 A learning automaton is optimal if $p_b(t) \rightarrow 1$ as $t \rightarrow \infty$, with probability 1.

This definition means that given enough time the learning automaton will eventually discover the right answer. But alas, there has yet to be a learning automaton that has achieved this [21]. This fact motivates the next definition; if the automaton can't pick the best action with probability of unity then it should be considered good if the probability of picking the best action can be made arbitrarily close to unity.

Definition 1.6 A learning automaton is said to be ϵ -optimal if the probability of choosing the best action can be made as close to unity as desired. More explicitly, if there is an internal parameter n , such that for all $\epsilon > 0$, $\delta > 0$, there exists $n_0 > 0$, and a $t_0 < \infty$ such that $\Pr [| p_b(t) - 1 | < \epsilon] > 1 - \delta$ for all $t \geq t_0$ and for all $n > n_0$, then the scheme is ϵ -optimal [11].⁶

More specifically, Definition 1.6 says that given enough time and given a large enough internal parameter n (typically something akin to the number of internal states) the probability of picking the best action almost all of the time (that is, with a probability of less than ϵ) can be made as close to unity as desired. It is clear that an optimal algorithm is an ϵ -optimal algorithm; if a learning automaton is always right, it is right as often as one would desire.

Rather than take the point of view that the automaton must converge on the right answer, another approach to measure the performance of the

⁶In the literature there is another common definition for ϵ -optimality. A learning automata is said to be ϵ -optimal if for all $\epsilon > 0$,

$$\lim_{t \rightarrow \infty} \inf p_b(t) > 1 - \epsilon$$

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

learning automaton is more pragmatic. This approach is to consider the probability that the automaton is penalized rather than the probability of picking the best action. This makes sense in the case where the best action and other actions have similar probabilities of being penalized. For example consider the case of Table 3.

Table 3
A Three Action Environment

<u>Action</u>	<u>Probability of Reward</u>
1	0.2
2	0.9
3	0.901

The probability of an automaton being rewarded would change very little if the automaton selected action 2 or action 3; their values are so similar. However, from the point of view of optimal and ϵ -optimal criteria, choosing action 2 is as undesirable as choosing action 1; neither is the optimal action. The first definition in this pragmatic vein is expediency.

Definition 1.7 A learning automaton is expedient if $\lim_{n \rightarrow \infty} E [M(t)] < M_0$,

where M_0 is the probability of being penalized if all actions were chosen randomly, and $M(t)$ is the probability of being penalized given its state at time t .

A simplistic approach, when dealing with a stochastic environment, is to always pick each action with equal probability. If a learning automaton can eventually beat this simplistic automaton it is called expedient. A learning automaton is expedient if, as learning continues, the automaton is

eventually expected to have a smaller probability of being penalized than if each action was selected randomly with equal probability. Obviously, this is a minimal requirement for an automaton which is said to be capable of learning.

A more stringent requirement is that the automaton improve its performance in each iteration of the input - output cycle. This requirement is called absolute expediency.

Definition 1.8 A learning automaton is absolutely expedient⁷ if $E[M(t+1) | \mathbf{P}(t)] < M(t)$ where $M(t)$ is the expected penalty given the automaton's probability of picking each action, and $\mathbf{P}$ is the action probability vector such that $\mathbf{P}(t) = (p_1(t), p_2(t), \dots, p_r(t))$ with $p_i(t) = \Pr \{ \alpha(t) = \alpha_i \}$.

With the standard performance criteria now defined, a general overview of the varieties of automata can be given.

1.8 Varieties of Learning Automata

The varieties of learning automata have been reviewed by Lakshmivarahan [11], and by Narendra and Thathachar [21, 22]. The amount of literature is so extensive that only a rough classification will be outlined here.

Currently, existing learning automata can be divided into categories based on their learning strategy, T . The various classifications are illustrated in Figure 8 below.

⁷Note that by taking expectations on both sides of Definition 1.8 one gets $E[M(t+1)] < E[M(t)]$. This means that $E[M(t)]$ is strictly decreasing in stationary random environments [10], and so $M(t)$ is a Supermartingale.

Figure 8 Classification of Existing Learning Automata

The differences between stochastic and deterministic automata were presented in sections 1.5 and 1.6. The first learning automata were deterministic. They, along with the concept of ϵ -optimality, were developed by Tsetlin. The first automaton shown to be ϵ -optimal in restricted environments was the Tsetlin automaton illustrated in Figure 9, for $A = \{\alpha_1, \alpha_2\}$ and with N states per action. The output function for the automaton is simple: if it is in state 1 to N then it chooses action α_1 and if it is in state $N+1$ to $2N$ then it chooses action α_2 .

Intuitively, if the automaton ever gets N more penalties than rewards then it will cross over and pick a new action. If it is rewarded, it will work itself deeper and deeper to one side and it subsequently takes even more penalties before the automaton will start choosing the other action. Since state 1 and

state $N+1$ are the deepest in memory they are referred to as the most internal states or end states.

If action is rewarded.

If action is penalized.

Figure 9. State Transitions for the Tsetlin's Automaton

It is clear that each state is recurrent. If action 1 is penalized N times in a row and then action 2 is rewarded N times in a row the state will work its way from 1 to state $N+1$. In a similar manner it will work itself back. The automaton is also aperiodic since it can loop in states 1 and $N+1$ an arbitrary number times. Hence it is ergodic and so the limiting distribution is independent of the initial distribution and unique. Let $c_i = 1-d_i$ for $i = 1, 2$. Hence c_i is the probability of an action i being penalized by the environment. The automaton can be shown to be ϵ -optimal only when $\text{Min}\{c_1, c_2\} \leq 0.5$ [21]. The expected penalty for Tsetlin's automaton is:

$$\lim_{t \rightarrow \infty} E[M(t)] = \frac{\frac{1}{c_1^{N-1}} \frac{c_1^N - d_1^N}{c_1 - d_1} + \frac{1}{c_2^{N-1}} \frac{c_2^N - d_2^N}{c_2 - d_2}}{\frac{1}{c_1^N} \frac{c_1^N - d_1^N}{c_1 - d_1} + \frac{1}{c_2^N} \frac{c_2^N - d_2^N}{c_2 - d_2}} \quad (1.6)$$

The Tsetlin automaton is deterministic. For an example of a stochastic automaton consider the Krylov automaton illustrated in Figure 10. The output function is identical to that of the Tsetlin automaton: if the machine is in states 1 to N it chooses α_1 and if it is in states N+1 to 2N it chooses α_2 . As can be seen from Figure 10, if the automaton is rewarded it reacts like the Tsetlin automaton does. However if it is penalized then it will move in either direction with probability 0.5.

Interestingly enough, this minor modification is enough to make the Krylov automaton ϵ -optimal in all environments. The expected penalty in the limit of time has been shown to be the following:

$$\lim_{t \rightarrow \infty} E[M(t)] = \frac{\frac{1}{\lambda_1^{N-1}} \frac{\lambda_1^N - 1}{\lambda_1 - 1} + \frac{1}{\lambda_2^{N-1}} \frac{\lambda_2^N - 1}{\lambda_2 - 1}}{\frac{1}{c_1^N \lambda_1^{N-1}} \frac{\lambda_1^N - 1}{\lambda_1 - 1} + \frac{1}{c_2^N \lambda_2^{N-1}} \frac{\lambda_2^N - 1}{\lambda_2 - 1}}$$

$$\text{where } \lambda_i = \frac{c_i}{2 - c_i} \quad i = 1, 2 \quad (1.7)$$

As N is increased it can be shown [21] that (1.7) tends to $\text{Min}\{c_1, c_2\}$. Both Tsetlin and Krylov's automata are similar in the sense that their probability of a state transition is not a function of time. In this sense they are fixed structure automata. Since Krylov's automaton is also stochastic it is called a fixed structure stochastic automaton (FSSA).

If action is rewarded.

If action is penalized.

Figure 10 State Transitions for the Krylov Automaton

1.9 Variable Structure Stochastic Automata

While the theory and methods of designing and analysing fixed structure automata are straightforward, the difficulty arises in the complexity of the equations encountered. This point is illustrated with equations (1.6) and (1.7). Varshavskii and Vorontsova first put forth the idea of variable structure stochastic automata (VSSA). For these automata, the learning process is generalised so that the state transition probabilities and the action selecting probabilities evolved with time [21]. The automaton is simplified in the sense that each state now corresponds uniquely to a particular action. Hence the G function always maps q_i onto α_i . Since this is characteristic of the whole class of automata, the G function is dropped from the notation. As well, the set of states is dropped. What remains is the set of actions (or output from the automaton), the set of inputs (or input to the automaton) and a learning algorithm T . The learning algorithm operates on a probability vector $\mathbf{P}(t) = (p_1(t), p_2(t), \dots, p_r(t))$ where $p_i(t) = \Pr \{ \alpha(t) = \alpha_i \}$, and 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 characterised by the method of updating the probability of choosing the actions [11,14,21,22]. This probability distribution over the state of actions is called the probability vector⁸. 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 automaton is in its end state when one of its components is unity.

Definition 1.9 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 $T : [0, 1]^r \times B \rightarrow [0, 1]^r$ is a learning algorithm such that $T(\mathbf{P}(t), \beta(t)) = \mathbf{P}(t+1)$, where $\mathbf{P}$ is the action probability vector such that $\mathbf{P}(t) = (p_1(t), p_2(t), \dots, p_r(t))$ with $p_i(t) = \Pr \{ \alpha(t) = \alpha_i \}$.

In general, VSSA are analysed from the point of view that their probability of choosing an action at a given time represents a discrete Markovian process. As in the case of FSSA, 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 [14, 22, 25, 31]. 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 [21,22,29,30,31,44]. Ergodic VSSA can go into and out of any state an unlimited number of

⁸In the literature this vector is called the action probability vector. Throughout this thesis, for the sake of simplicity, it will often be referred to as the probability vector.

times. Since they are ergodic, 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 may be 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 input is necessary for the probabilities to be updated by the automaton. Their meanings are explained in Table 4.

Table 4

Varieties of Updating Schemes

<u>Symbol</u>	<u>Full Form</u>	<u>Meaning</u>
RI	Reward-Inaction	Updates probabilities when automata are rewarded, penalties are ignored.
IP	Inaction-Penalty	Updates probabilities when automata are penalized, rewards are ignored.
RP	Reward-Penalty	Updates probabilities when automata are rewarded or penalized.

For an example of VSSA consider the linear schemes designated L_{RP} , L_{RI} , and L_{IP} . The updating scheme is linear in the sense that whenever the probability of choosing an action is decreased it is achieved using a linear

function. The two action varieties are presented in Table 5.

Table 5--A Comparison of Updating
Rules for Various Two Action Learning Automata

Algorithm LRP:		
$p_1(t+1)$	$= p_1(t) + \lambda(1 - p_1(t))$	if α_1 is chosen and rewarded, or α_2 is chosen and penalized
	$= p_1(t) (1 - \lambda)$	if α_2 is chosen and rewarded, or α_1 is chosen and penalized
Algorithm LRI:		
$p_1(t+1)$	$= p_1(t) + \lambda(1 - p_1(t))$	if α_1 is chosen and rewarded
	$= p_1(t) (1 - \lambda)$	if α_2 is chosen and rewarded
	$= p_1(t)$	if the automaton was penalized.
Algorithm LIP:		
$p_1(t+1)$	$= p_1(t) + \lambda(1 - p_1(t))$	if α_2 is chosen and penalized
	$= p_1(t) (1 - \lambda)$	if α_1 is chosen and penalized
	$= p_1(t)$	if the automaton was rewarded.

Interestingly enough, the algorithms have different properties. The LRP and LIP are ergodic and the LRI is absorbing. To see that LRI is absorbing consider the case when $p_1(t)$ becomes unity. If choosing α_1 is rewarded then, from Table 4, $p_1(t+1) = 1 + \lambda(1-1) = 1$. On the other hand if α_1 is penalized then the probabilities are not updated since the scheme is a Reward-Inaction scheme.

An examination of the LRP and LIP show that when $p_1(t)$ becomes unity, the value will no longer be unity the next time the automaton is penalized. Interestingly enough, the LRI scheme is ϵ -optimal and the LRP and LIP are not [31]. However since the LRP and the LIP are ergodic, they have the flexibility of being appropriate for both stationary and non-

stationary environments. Since the LRI scheme is absorbing, it would be inappropriate

for non-stationary environments. If an action is optimal only for a short period of time, there is the risk that the automaton might converge to it and get locked into it even after it is no longer optimal.

1.10 Applications of LA

Learning algorithms are useful whenever complete knowledge about an environment is unknown, expensive to obtain or impossible to quantify. Thus they have found applications in various fields including game playing [1,11,13] pattern recognition [22, 36], and object partitioning [32, 33]. Learning automata are also useful when the environment with which they interact varies from implementation to implementation and thus they have found applications in priority assignments in a queuing system [15, 16], and the routing of telephone calls [23, 24]. To give an idea of their utility three applications will be briefly considered: the routing of telephone calls, priority assignments in a queue and object partitioning.

1.10.1 Telephone Traffic Routing

When telephones were first developed it was soon realized that rather than have a line from each telephone to every other telephone, it would take much less wiring if each telephone was connected to a central office, and that connections between telephones would take place at the central office. A further saving is realized if instead of having one central office, a central office is associated with each community, and the central offices themselves are linked together by trunks. Since it is unlikely that everyone in one

community would simultaneously pick up their telephones and want to speak to people outside their community, there are generally less trunks between central offices than there are telephones connected to a central office. Central offices are typically far apart, and so any reduction in the number of trunks tends to be a large saving in the amount of wire that is used. Abstracting the central offices as nodes and the trunks that join them as vertices, the problems becomes that of optimizing traffic flow in a network.

The problem is complicated by the fact that telephone usage is random, generally abstracted as a Poisson process. Consider the example from [21] illustrated in Figure 11.

Figure 11 Example of a Network for Traffic Flow

Telephone traffic arrives at node 1 with a Poisson distribution and a mean of 5 calls per minute. If traffic was routed selectively through trunks (1,2) and (2,3), traffic could be blocked at node 2 because (2,3) has a lower capacity than (1,2). The extra capacity of route (1,4) (4,3) would go unused. Likewise when routing through (1,4) (4,3) traffic would be blocked at node 4 because (4,3) has a lower capacity than (1,4) and route (1,2) (2,3) would go

unused. Clearly a mixed strategy would be best. An automaton can be set up as follows:

- 1] The set of actions is: { pick route (1,2), pick route (1,4) }.
- 2] The automaton is rewarded only if the call is not blocked.

Experimentally, when an LRI learning scheme is used, after 5000 iterations the automaton had a mixed strategy that resulted with a blocking probability of 0.186 when the theoretically optimum blocking was 0.150 [21].

Recall that the traffic flow varies with the time of day. To do statistical studies would be expensive and the studies would have to be repeated as communities develop around metropolitan centres, and patterns of traffic flow change. A learning automaton provides an excellent tool because it would dynamically adopt the optimal strategy.

1.10.2 Priority Assignments in a Queue

Another application of learning automata is priority assignment in a queue. It is well known that the mean processing time of a queue of jobs can be decreased by putting the smallest jobs first [21]. For example, Figure 12 illustrates four jobs: job 1 takes one second, job 2 takes two seconds, and so on. Even though the total time taken to process all jobs remain the same (that is, ten seconds), if the jobs are arranged in the descending order of their execution time the mean processing time is fifty percent slower than if they are arranged in the reverse order. After six seconds three jobs have been completed in queue A and only one job has been completed in queue B.

Imagine that the processor polls two queues. If the tasks in queue A have a shorter mean service time than those of queue B then a learning

automaton can be used to assign priorities and hence optimize the scheduling of service. The automaton can be set up as follows: as each job is processed, the processor keeps a running estimate of the mean service time. In this case the automaton has the following specifications:

1] The set of actions is: pick from queue A, or pick from queue B.

2] The automaton is rewarded only if the processing time of that job is less then the estimate of the mean service time.

Figure 12 Jobs Size and Mean Processing Time

If the arrival times are Poissonly distributed and the processing times are exponentially distributed then it has been shown that the L_{RI} automaton will converge, in an ϵ -optimal fashion, to picking the queue with the shortest mean processing time [16]. The fact that the mean processing time does not have to be known beforehand, makes this scheme particularly attractive.

1.10.3 Object Partitioning

Learning automata have also been used to find solutions to the object

partitioning problem. Basically, given a set of objects, they are to be put in equal sized groups so that the objects that are most frequently accessed together are in the same group. An example of this application is a storage room with many shelves. Since a hammer and nails are frequently used together, the time spent searching for equipment could be improved if nails were kept with the hammers rather than beside the screwdrivers. A more realistic application would be that of storing records that are frequently used together on the same block or track of a disk. Similarly, for a distributed database, it would be desirable to have frequently co-accessed data in the same physical location [33, 34]. The automaton tackling the object partitions problem has the following implementation:

1) The set of actions is the possible partitions.

2) The automaton is rewarded only if the two objects requested are in the same partition.

Simulations using Tsetlin and Krinsky automata show that this method is an order of magnitude faster than the previously known best method [33, 34]. The only problem is that as the number of objects increases the number of partitions, (and hence the number of actions) increases exponentially.

A modification of Tsetlin automata, called the Object Migrating automaton, simplifies this. This is illustrated in Table 6. In this case, there are six objects: $o_1, o_2, o_3, o_4, o_5, o_6$, and the task is to divide the objects into two groups. There are two numbers associated with each object. The first is the partition number which specifies which partition the object is in. The second is its state index. With this index, the object migrating automaton, in one sense, mimics the operation of the Tsetlin automaton. If two objects are in the same partition and they are requested together, this partition is

rewarded by driving the objects deeper into their partition. This is achieved by increasing both of their state indices, up to a prespecified maximum. Upon being penalized, they move towards the boundary states by decreasing their state indices. If an object has zero for a state index then it is swapped with the object that has the lowest state index from another partition.

To illustrate this automaton, consider Table 6. The first panel in Table 6 shows that, initially, objects o_1 , o_4 and o_5 are in partition 1, and objects o_2 , o_3 , and o_6 are in partition 2. The changes between panels are printed in bold type. In the second panel objects o_1 and o_4 are requested and both are in partition 1, and so their state indices are increased from 7 to 8 and from 6 to 7 respectively. In the third panel, objects o_1 and o_6 are picked and since they are in different partitions, their state indices are decreased. Since object o_6 now has depth 0 it is swapped with the object with the lowest depth from partition 1, which in this case, is o_5 . The results of the swap is shown in the last panel of Table 6.

The object partitioning problem, the network flow problem and the queueing problem all have similar features. Statistical studies could be made for each particular situation to arrive at an optimal solution. Learning automata provide a general solution and they work for all cases, and yet they are relatively easier to implement. If the designer of the algorithm has an estimate of the optimal solution, then this can be incorporated in the initial vector and the automaton will fine tune this solution [30]. For example, in the telephone routing case, if route two has twice as many trunks as route one, then the automaton can be initialized to select route two twice as often as route one. The automaton will then learn if this is the best solution or if the ratio should be changed slightly.

Table 6--Simulation of Object Migration

Maximum Depth: 8 6 objects : 01,02,03,04,05,06
 Number of Partitions:2 Current Arrangement: (01,04,05),(02,03,06).

Initially:

Objects	01	02	03	04	05	06
Partition #:	1	2	2	1	1	2
State Index:	7	8	8	6	2	1

Objects requested : 01,04

In same partition $\Rightarrow$ reward, increase state indices by one:

Objects	01	02	03	04	05	06
Partition #:	1	2	2	1	1	2
State Index:	8	8	8	7	2	1

Objects requested : 01,06

In difference partitions $\Rightarrow$ penalize, decrease state indices by one:

Objects	01	02	03	04	05	06
Partition #:	1	2	2	1	1	2
State Index:	7	8	8	7	2	0

Since 06 index has reached 0 $\Rightarrow$ swap with the object which has the lowest state index from the other partition. In this case 05 is swapped to yield:

Objects	01	02	03	04	05	06
Partition #:	1	2	2	1	2	1
State Index:	7	8	8	7	1	1

With a basic overview of learning automata complete, a general method of improving the performance of VSSA will be introduced in the next chapter.

CHAPTER 2

DISCRETE LEARNING ALGORITHMS

2.1 Introduction

"As any programmer knows, there's no difference between theory and practice *in theory*, but often a great deal of difference between theory and practice *in practice*." (italics mine) [4]. Although whimsical, this statement inspires the motivation, and in fact the necessity, for considering discrete learning algorithms. The beauty and the brilliance of a discrete learning algorithm is that it does not ignore the limitations of practical implementations; this reality is used to an advantage.

This chapter describes initially how a VSSA is rendered discrete⁹. Next, the discrete learning algorithms that already exist will be considered. Finally, the advantages of discrete learning automata over their continuous counterparts will be discussed.

Chapter 1 described automata as a natural step towards a general solution to the problem of machine learning. VSSA evolved from FSSA as an attempt to simplify the analysis of the automata's properties [21]. 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

⁹ Here discrete refers to a set which has no accumulation points. The Bolzano Weierstrauss theorem states: Any infinite subset of a bounded set has an accumulation point. Hence to make the set $[0, 1]$ discrete is to make it finite [8]. This implies a decrease in the cardinality of the set.

interval $[0, 1]$. Rendering this probability space discrete is a general approach for improving VSSA [39]; 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. Such automata are discrete versions of their continuous counterparts. 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 varies with time. Discrete algorithms are linear if the probability values are equally spaced in the interval $[0, 1]$; otherwise, they are called non-linear [39]. Existing literature [31] uses the term 'discretized' in front of the name of a learning automaton to indicate the discrete version of a continuous VSSA [31].

2.2 Existing Discrete Automata

Without considering the discrete estimator algorithms described in Chapter 4, so far the theoretical properties of seven discrete algorithms have been described in the literature [27, 28, 31, 39]. Five of these algorithms are based on the two action linear schemes. All three variants, 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 counterparts.

The first results concerning discretized learning automata were largely experimental [39]. 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. To describe the DL_{RI} schem, as in the previous chapter, let p_1 is the probability the automaton chooses α_1 , and p_2 is the probability that the automaton

chooses α_2 . The term probability vector is used when referring to both p_1 and p_2 . Since $p_1 + p_2 = 1$, the automaton will be defined solely in terms of p_1 . Given an internal parameter¹⁰ n , the probability of choosing an action is updated using the algorithm presented below:

ALGORITHM DL_{RI}

Parameters n , a positive integer.

Initialization $p_1(0) = 0.5$
 $p_2(0) = 0.5$

Repeat:

Step 1: At time t pick $\alpha(t)$ according to probability distribution $P(t)$.

Step 2 $p_1(t+1) = \text{Min}\{1, p_1(t) + 1/n\}$ if action 1 is chosen and rewarded
 $= \text{Max}\{0, p_1(t) - 1/n\}$ if action 2 is chosen and rewarded
 $= p_1(t)$ if any action penalized
 $p_2(t+1) = 1 - p_1(t)$

End Repeat

END ALGORITHM DL_{RI}

The first series of experiments with discrete automata were reported by Thathachar and Oommen [39]. For the experiments reported, c_1 was always 0.8; and c_2 varied from 0.4 to 0.7. The accuracy, or number of times it converged correctly (designated by $E[p_1(\infty)]$), using various values of n was calculated for the DL_{RI}, Tsetlin, and Krinsky automata. Generally speaking, the DL_{RI} scheme was always more accurate. Through simulation

¹⁰For the two action case, n is generally assumed to be even, however this is not absolutely necessary.

runs, the DL_{RI} scheme was also found to reach the end state quicker. Some of the results are summarized in Table 7 [39] in which Iter. refers to the number of iterations for the scheme to converge to an absorbing state.

Table 7

The DL_{RI} Compared with the Tsetlin and Krinsky Automata

c_1	c_2	n	DL_{RI}		Tsetlin		Krinsky	
			$E[p_1(\infty)]$	Iter.	$E[p_1(\infty)]$	Iter.	$E[p_1(\infty)]$	Iter.
0.4	0.8	4	0.84	6	0.80	30	0.80	16
0.4	0.8	10	0.95	16	0.95	68	0.97	56
0.7	0.8	4	0.65	11	0.57	175	0.57	12
0.7	0.8	10	0.73	34	0.60	2000	0.66	50

Apart from simulation results, Oommen and Hansen proved that the DL_{RI} is ϵ -optimal [27]. The L_{RI} algorithm was compared to the DL_{RI} scheme [27]. Since the accuracy can be calculated theoretically for both automata, they were put in an environment and given parameters so that they would have the same lower bound for accuracy. The two algorithms were compared to see which would reach this bound first. In one experiment with $c_1 = 0.2$ and $c_2 = 0.6$, it was observed that the DL_{RI} reached an accuracy of 99 % with 125 iterations, whereas the L_{RI} took 135 iterations. The other experiment showed that for a fixed number of iterations, 240, the DL_{RI} scheme converged to the optimal action more often than the L_{RI} scheme.

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 [27]. This is particularly significant because it was the first Inaction-Penalty

scheme to be proven ϵ -optimal. Oommen and Christensen also proved that the Discrete L_{RP} is ϵ -optimal in certain environments [28]. A version of the DL_{RP} scheme with absorbing barriers was shown to be ϵ -optimal in all environments [28]. These results are particularly significant because their continuous counterparts are not ϵ -optimal in any environment ! Discrete automata are also ϵ -optimal outside the linear family of schemes. For the sake of completeness, two such ϵ -optimal schemes will be mentioned. Oommen proved a non-linear discrete algorithm ϵ -optimal [31], and Oommen and Christensen discovered a modified discrete L_{RP} scheme that was both ergodic and always ϵ -optimal [28].

Rather than describe the other variants in detail, in this chapter only the discrete L_{RI} has been presented formally. The DL_{IP} , and DL_{RP} are identical except for the conditions under which probabilities are changed. The updating rule for each variation is presented in Table 8. The conditions when automata are ϵ -optimal are presented in Table 9. In both tables, the columns represent the three updating strategies: RI, IP and RP. The rows represent the various types of probability spaces encountered: 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 absorbing barriers added 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.

Recall from section 1.9 that the L_{RI} scheme is already absorbing; the same argument reveals that the discretized version is also 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.

Table--8 A Comparison of Updating Rules For Various Types Of Learning Automata

	Reward-Inaction	Inaction Penalty	Reward Penalty
Discrete	Algorithm DLR:	Algorithm DL _{IP} :	Algorithm DL _{RP} :
α_1 rewarded	$p_1(t+1) = \text{Min}\{1, p_1(t) + 1/n\}$	$p_1(t+1) = p_1(t)$	$p_1(n+1) = \text{Min}\{1, p_1(t) + 1/n\}$
α_2 rewarded	$= \text{Max}\{0, p_1(t) - 1/n\}$	$= p_1(t)$	$= \text{Max}\{0, p_1(t) - 1/n\}$
α_1 penalized	$= p_1(t)$	$= \text{Max}\{0, p_1(t) - 1/n\}$	$= \text{Max}\{0, p_1(t) - 1/n\}$
α_2 penalized	$= p_1(t)$	$= \text{Min}\{1, p_1(t) + 1/n\}$	$= \text{Min}\{1, p_1(t) + 1/n\}$
Absorbing		Algorithm ADL _{IP} :	Algorithm AD ^L _{RP} :
α_1 rewarded		$p_1(t+1) = p_1(t)$	$p_1(n+1) = \text{Min}\{1, p_1(t) + 1/n\}$
α_2 rewarded		$= p_1(t)$	$= \text{Max}\{0, p_1(t) - 1/n\}$
α_1 penalized		$= \text{Max}\{0, p_1(t) - 1/n\}$	$= \text{Max}\{0, p_1(t) - 1/n\}$
α_2 penalized		$= \text{Min}\{1, p_1(t) + 1/n\}$	$= \text{Min}\{1, p_1(t) + 1/n\}$
absorbing condition		$= 0 \text{ if } p_1(t) = 0$ $= 1 \text{ if } p_1(t) = 1$	$= 0 \text{ if } p_1(t) = 0$ $= 1 \text{ if } p_1(t) = 1$
Here p_1 is the probability of selecting α_1 and p_2 is the probability of selecting α_2			

Table 9 --A Comparison of Asymptotic Properties of Linear VSSA

Reward-Inaction		Inaction Penalty	Reward Penalty
<u>Continuous</u>	Algorithm LRI: ϵ optimal in all environments	Algorithm LIP: **	Algorithm LRP: $c_2 / (c_1 + c_2)$
<u>Discrete</u>	Algorithm LRI: ϵ optimal in all environments	Algorithm DLP: $c_2 / (c_1 + c_2)$	Algorithm DLRP: ϵ optimal if minimum penalty prob. < 0.5
<u>Absorbing</u>		Algorithm ADLP: ϵ optimal in all environments	Algorithm ADLRP: ϵ optimal in all environments

** Could not find asymptotic properties of this algorithm.

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. This is shown in Table 10.

Table 10
Comparison of the Updating Rules for L_{RI} and DL_{RI}

	<u>L_{RI} (with $\lambda = 0.5$)</u>	<u>DL_{RI} (with $n = 4$)</u>
Initially	$p_1 = 0.5$ $p_2 = 0.5$	$p_1 = 0.5$ $p_2 = 0.5$
α_1 rewarded	$p_1 = 0.75$ $p_2 = 0.25$	$p_1 = 0.75$ $p_2 = 0.25$
α_2 rewarded	$p_1 = 0.375$ $p_2 = 0.625$	$p_1 = 0.5$ $p_2 = 0.5$

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. If the environment is stationary, then the DL_{RI} is preferred because it is not biased towards more recent responses. In this case, the probability will increase by $1/n$ per reward anywhere in the probability space except at the end states. Its property can be summarized informally by saying the amount the probability vector changes depends on its value. As the probability of choosing α_1 gets

closer to zero, the magnitude of the increase in p_1 will get bigger, because of the factor $(1-p_1)$. Therefore, if the action has been penalized recently, it will have a greater change in probability than if it has been rewarded recently for in this case the quantity $(1-p_1)$ is smaller

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. This is illustrated in Table 11, when $n = 10$, and $p_1 = 0.7$, and $p_2 = 0.3$.

Table 11 Comparison of DL_{RI} and DL_{IP} Automata

<u>DL_{RI}</u>	<u>DL_{IP}</u>
Initially	Initially
$p_1 = 0.7$ $p_2 = 0.3$	$p_1 = 0.7$ $p_2 = 0.3$
α_1 rewarded	α_2 penalized
$p_1 = 0.8$ $p_2 = 0.2$	$p_1 = 0.8$ $p_2 = 0.2$
α_2 penalized	α_1 rewarded
$p_1 = 0.8$ $p_2 = 0.2$	$p_1 = 0.8$ $p_2 = 0.2$

When differing actions are chosen and complementary responses are received from the environment, the two automata update their probability vectors identically. Increasing p_1 by 0.1 in the DL_{RI} case has the same effect as decreasing p_2 by 0.1 in the DL_{IP} case. Because the sum of probabilities has to equal unity, if the component of one probability vector for an automaton is increased the other must be decreased.

If the DL_{IP} automaton could be implemented using the DL_{RI}

automaton, then the ϵ -optimality of one could be used to prove the ϵ -optimality of the other. The amazing fact, as seen in Table 9, 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 [28]. A similar observation can be made for the LRP and the absorbing DL_{RP} . 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.

2.3 Motivation for Discretization

Probably the biggest limitation of learning automata is their slow rate of convergence [40]. There seems to be a trade off here. 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 converge and to eliminate the assumption that the random number generator could generate real numbers with arbitrary precision [31]. Each of these points will be dealt with in turn. A hypothetical conversion from a continuous to a discrete probability space is illustrated in Figure 13.

Once the optimal action has been determined, say at time t , and the corresponding action probability is relatively close to unity, the discrete automata increases the probability of choosing that action to the value of

unity directly, rather than approach the value unity asymptotically. Figure 13 illustrates that for the L_{RI} case, if the probability of choosing the optimal action is at 0.98, and the automaton is rewarded two times, then at time $t + 2$ the probability of choosing the optimal action will be close to unity. The value will depend on the parameters of the scheme. However, in the discretized case, if the probability space is divided into intervals of width 0.01, the probability of choosing the optimal action will be exactly unity.

Figure 13 A Comparison of the Probability of Choosing the Optimal Action for a Continuous and a Discretized Learning Algorithm

If the quantity p_1 goes directly from 0.98 to 1.0, this will change the mean probability of the automaton getting a reward from:

$$0.98 d_1 + 0.02 d_2 \quad \text{to} \quad d_1$$

Consider the worst case scenario in which one action is always rewarded and the other action is always penalized. This change corresponds to at most a 2 % change in the probability of being rewarded. If this 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. In this way, if the algorithm is very close to an end state, the minimum step size forces the automaton to the end 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 [37]. As well, discrete Markov processes have properties that are not true for general Markov processes [37].

To give an explicit example, consider the LRI algorithm. When the LRI scheme starts at any point other than its end point, it approaches its end point asymptotically, and will never reach it in finite time. However when implemented on a machine with finite architecture, as the probability gets closer to unity, round-off error will cause the value to become unity artificially [31]. Thus the automaton has ended up in its final state in a finite amount of time ! With this in mind, proofs of convergence in continuous spaces may not be applicable to a finite state machine. This point is demonstrated by the fact that the existing proofs of convergence for discrete algorithms are different from the proofs of their continuous counterparts (compare [27] and [31] to [21]).

Another benefit of discretizing the probability of choosing an action is that it reduces the requirements on the system's random number generators

[28, 31]. This is important since VSSA use a random number generator in its implementation [39]. 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. This is due to the fact that the values are truncated after a fixed number of decimal places due to the finite precision allowed by the finite memory representations of floating point numbers. As well, the algorithms used for generating random numbers are typically integer methods [10] rather than floating point methods.

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 one iteration of the task. It also reduces the amount of memory needed. Practically speaking, the implementation of integer addition and subtraction is quicker than integer multiplication or division on a digital computer. For a typical microprocessor, addition is five to nine times faster than either multiplication or division.¹¹ The amount of memory used for a real number is more than that required for an integer, because in the former case both a mantissa and an exponent value are stored. 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.

Another practical consideration involves the difference between

¹¹ This information is based on the number of clock cycles it takes for the Motorola MC68008, MC68010, MC68020 series of microprocessors to perform unsigned integer addition, subtraction, multiplication, and division [17, 18].

precision and accuracy. Precision is the number of decimal places the value has; accuracy is how close the measured value is to the real value [26]. If the actual probability of reward is 0.5, and two estimates are 0.31415 and 0.4, the first is more precise and the second is more accurate.

The difference between precision and accuracy has an impact on speed. In general, there is a speed-memory trade off with computer algorithms. If more memory is used, the algorithm will run faster. Learning automata are an exception. For a discrete stochastic automaton, more memory implies more states, which means more precise estimates of the probability vector. However from the previous considerations, precision does not imply accuracy. In a manner analogous to the use of double precision arithmetic on a computer, when only single precision is necessary, the unwarranted precision can slow down the algorithm. Since both discrete and continuous automata make use of an internal parameter, the precision of the probability vector should be linked to this internal parameter. If an integer range is used to represent the discrete probability space, the compiler of languages such as Pascal, or Modula will pick the most efficient representation at compile time (such as a bit, a byte or a multiple word representation). However if a probability is represented as a real number, the choices of representation are more restricted, typically single or double precision. As well this precision must be prespecified by the programmer.

In summary, discrete versions lead, quite naturally, to the use of integers for keeping track of how many multiples of $1/n$ the probabilities are. Since many methods of generating random numbers are integer methods [10], discrete algorithms are natural candidates for using only integer arithmetic. Since integer arithmetic is faster than real arithmetic, this fact

would also tends to speed up computation time. Integer arithmetic based algorithms have the advantage that they are easier to implement in a low level language such as assembly language or Forth which are based on integer numbers only.

With a general description of discrete VSSA complete, a new class of automata that are good candidates for being discretized will be considered next. These are the estimator algorithms.

CHAPTER 3

ESTIMATOR ALGORITHMS

3.1 Characterisation and Motivation

A very recent approach to improving the rate of convergence of learning automata was introduced by Thathachar and Sastry using the so-called estimator algorithms [19, 38, 40-43]. The motivation for these algorithms is simple. If a naïve individual was asked to perform the task of picking the most frequently rewarded action in a stochastic environment, this person would most likely try each action a number of times and estimate the probability of reward for each action. Actions that received low estimates of reward probabilities, would be eliminated first. Finally, when enough confidence has been built up, an optimal action would be decided upon. This is precisely the idea that inspires the family of estimator algorithms.

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. The major difference between the two vectors is that the components of the probability vector must sum to exactly unity because it is a probability distribution over a set of possible actions. However, it is possible to have an estimate vector whose sum of components exceeds unity. For example, let the first action be 'take a bus to school', and let the second action be 'walk

to school'. A penalty will correspond to the event of 'being late'. A possible value for the probability vector is (0.6, 0.4) and a possible value for the estimate vector is (0.9, 0.85). That is, there is a probability of 0.6 that you will take the bus and there is probability of 0.9 that you will arrive on time given that you have taken the bus.

Table 12 compares the structure of a non-estimator algorithm to an estimator algorithm. Typically, non-estimator algorithms update the probability of choosing an action based directly on the feedback obtained from the environment. If an action was rewarded, then the probability of choosing that action is increased. It is in this respect that estimator algorithms are fundamentally different. When an action is selected, based on the probability vector, and feedback from the environment is obtained, an estimator algorithm first uses this information to update the estimate vector. The probability vector is updated based on the estimate vector and the current response from the environment, rather than merely on feedback 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 estimator algorithms [40]. 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 LRI scheme [38, 40, 43].

3.2 The Pursuit Algorithm

Pursuit Algorithms are a subclass of estimator algorithms[19, 41, 43]. These algorithms are characterised by the fact that they pursue the

optimal action. This is done in two steps. When feedback is obtained, the automaton calculates which action has the highest estimate for the probability of reward. The pursuit algorithm increases the action probability whose current estimate is the maximum [43].

Table 12 Comparison of the Basic Features of Estimator
and Non-estimator Algorithms

<u>Non -estimator Algorithms</u>	
Vectors:	$\mathbf{P}(t)$, the probability vector.
Updating:	Based directly on feedback from the environment.
<u>Estimator Algorithms:</u>	
Vectors:	$\mathbf{P}(t)$, the probability vector. $\mathbf{D}'(t)$, the estimate vector.
Updating:	Based on the estimate vector and the response of the environment.

The continuous Pursuit Algorithm [43] involves three steps. The first step is to pick the action $\alpha(t)$ based on the probability distribution $\mathbf{P}(t)$. The second step is to update $\mathbf{P}(t)$ based on feedback from the environment as illustrated below:

$$\begin{aligned} \mathbf{P}(t+1) &= \mathbf{P}(t) && \text{if } \beta(t) = 1 \\ &= (1 - \lambda) \mathbf{P}(t) + \lambda \mathbf{e}_m && \text{if } \beta(t) = 0 \end{aligned} \quad (3.1)$$

The vector $\mathbf{e}_m$ is the unit vector in the direction of the largest component of $\mathbf{D}'(t)$. Note that the magnitude of the update is based on a parameter λ , a number strictly between zero and one. The scheme is an RI scheme, and

thus if the most recent action was penalised then the probability vector $\mathbf{P}$ stays the same. Once an action is rewarded, the vector $\mathbf{P}(t)$ is updated. The new value is a weighted average of $1-\lambda$ times the old value plus λ times the unit vector which reflects the optimal action based on the current estimates. In terms of vectors, (3.1) illustrates that the old probability vector is moved in the direction of the current maximal estimate of reward.

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

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

m : the maximal component of $\mathbf{D}'(t)$, $d'_m(t) = \mathbf{Max}_i \{d'_i(t)\}$;

$\mathbf{e}_m$: 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 $1 \leq i \leq r$.

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

Initialization $p_i(0) = 1/r$ for $1 \leq i \leq r$

Initialize $\mathbf{D}'(t)$ by picking each action a small number of times.

Repeat:

Step 1: At time t pick $\alpha(t)$ according to probability distribution $\mathbf{P}(t)$.

Step 2: Update $\mathbf{P}(t)$ according to the following:

$$\begin{aligned} \mathbf{P}(t+1) &= \mathbf{P}(t) && \text{if } \beta(t) = 1 \\ &= (1 - \lambda) \mathbf{P}(t) + \lambda \mathbf{e}_m && \text{if } \beta(t) = 0 \end{aligned}$$

Step 3: Update $\mathbf{D}'(t)$ according to the following:

$$\begin{aligned} \text{if } \alpha(t) = \alpha_j, \quad W_j(t+1) &= W_j(t) + (1 - \beta(t)) \\ Z_j(t+1) &= Z_j(t) + 1 \\ d'_j(t+1) &= W_j(t+1) / Z_j(t+1) \end{aligned}$$

$$\begin{aligned} \text{for all } i \neq j \quad W_i(t+1) &= W_i(t) \\ Z_i(t+1) &= Z_i(t) \\ d'_i(t+1) &= d'_i(t). \end{aligned}$$

End Repeat

END ALGORITHM Pursuit

The pursuit algorithm is similar in design to the L_{RI} algorithm, except that whereas the L_{RI} algorithm moves $\mathbf{P}(t)$ in the direction of the most recently rewarded action, the pursuit algorithm moves $\mathbf{P}(t)$ in the direction of the action which possesses the highest estimate of reward. Both algorithms approach their end points (when $\mathbf{P}(t)$ has all but one non-zero component) asymptotically. This is because the action probability to which the scheme is converging is increased by a quantity proportional to the sum

of the remaining action probabilities. As this probability gets closer to converging, the magnitude of the remaining components decreases, and hence the the highest action probability is increased by a correspondingly smaller amount.

Although in theory the L_{RI} scheme reaches its absorbing state only in the limit, the finite architecture of digital computers will eventually round off the value of the probability vector to either one or zero. When this happens the algorithm will always pick that action and no combination of reward and penalties can change this. However in the case of the Pursuit Algorithm, if the action it has converged to receives a large amount of penalties, the result may eventually reduce the estimate of the reward probability so that it is no longer the maximum. This allows the scheme to leave this particular end state and pursue another action. The pursuit algorithm will converge to the optimal action, but if the action ceases to be optimal by the algorithm's estimate, then the algorithm will pursue the new optimal action. In summary, for applications on finite machines, the L_{RI} is absorbing and the pursuit algorithm may not be.

Thathachar and Sastry [43] 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. Table 13 lists the results two environments, each with ten actions. In the results, the largest value of λ that gave 0 errors in 25 experiments is reported for the Pursuit algorithm and for the L_{RI}. The Pursuit algorithm was seven times faster in environment A and five times fast in environment B. When λ was adjusted so that the L_{RI} scheme took approximately the same number of

iterations to converge as the Pursuit Algorithm, the LRI converged inaccurately, 8 times out of 25. Like the LRI scheme, the Pursuit algorithm has been shown to be ϵ -optimal [43].

Table 13
Performance of the Pursuit Algorithm and LRI

Environment	Algorithm	λ	# of Iterations	Errors
Envt A	Pursuit	0.1	155	0
	LRI	0.02	1176	0
	LRI	0.1	182	8
Envt B	Pursuit	0.008	909	0
	LRI	0.01	4402	0
	LRI	0.04	620	8

Reward probabilities are:

Envt A	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
Envt B	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

3.3 The TS Estimator Algorithm¹²

The TS Estimator algorithm (TSEA) 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

¹²Thathachar and Sastry refer to this algorithm as just an estimator algorithm. However this thesis must distinguish it from the other estimator algorithms, and so we will call it the TS estimator algorithm.

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 $\mathbf{D}'(t)$, and the action probability vector $\mathbf{P}(t)$, as seen in (3.2). This algorithm is formally described as below.

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

ALGORITHM TS Estimator

Parameters

$\lambda, m, \mathbf{e}_m, W_i(t), Z_i(t)$ are the same as the Pursuit Algorithm.

$S_{ij}(t)$: an indicator function,

$$\begin{aligned} S_{ij}(t) &= 1 \text{ if } d'_i(t) > d'_j(t) \\ &= 0 \text{ if } d'_i(t) \leq d'_j(t). \end{aligned}$$

f : is any monotonic, increasing function¹³,

$f: [0, 1] \rightarrow [0, 1]$, satisfying $f(0) = 0$ and $f(1) = 1$.

Initialization $p_i(0) = 1/r$ for $1 \leq i \leq r$.

Initialize $\mathbf{D}'(t)$ by picking each action a small number of times.

¹³The typical family of functions that are used here are $f(x) = x^n$, where n is one of $\{ \dots 1/4, 1/3, 1/2, 1, 2, 3, 4, \dots \}$.

Repeat:

Step 1: Same as Pursuit algorithm.

Step 2: Let $\alpha(t) = \alpha_j$, update $\mathbf{P}(t)$ according to the following:

for the i^{th} component,

$$p_i(t+1) = p_i(t) + \lambda \sum_{j \neq i} \left[f(d'_i(t) - d'_j(t)) \left(S_{ij}(t) p_i(t) + S_{ji}(t) \frac{(1-p_i(t)) p_i(t)}{r-1} \right) \right]$$

for all other j , such that $j \neq i$.

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

Step 3: Same as Pursuit algorithm.

End Repeat

END ALGORITHM TS Estimator

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

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

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

$$p_j(t+1) = p_j(t) - \lambda \left[f(d'_i(t) - d'_j(t)) (p_i(t)) \right] \quad (3.4)$$

It is worth noting that if for all x , $f(x) = 1$ then (3.4) reduces to the pursuit algorithm above. 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.

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 (3.3). In other words, if there are actions with estimates bigger than the i^{th} action's estimate, their probability of being chosen will increase. This is just as Thathachar and Sastry point out [42]: though one action is rewarded, other actions may increase their probability of being chosen. As a matter of fact, if the i^{th} 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 (3.3). Even if the i^{th} action has the smallest estimate of being rewarded, the total increase of all the other $p_j(t)$, $j \neq i$, 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 (3.3) and the $p_j(t)$ factor in (3.4) 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,

$p_j(t)$, $j \neq i$, will be updated according to (3.4). 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. Hence the scheme need not be absorbing.

This algorithm has also been shown to be ϵ -optimal [42]. Because of the unique possibility of decreasing an action that has just been rewarded, the TSEA has no analogous non-estimator counterpart. Simulation comparisons with the LRI 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 TS estimator algorithm converges almost an order of magnitude quicker [42] as can be seen from Table 14.

Two methods of increasing the rate of convergence of learning automata have been discussed so far:

- i) discretizing the probability space, and,
- ii) using estimates of the reward probabilities.

A hybrid of these two methods should be considered; this is the primary contribution of this thesis.

Table 14
Performance of the TS Estimator Algorithm and LRI

Environment	Algorithm	λ	# of Iterations	Errors
Envt A	TSE	1.0	167	0
	LRI	0.02	1176	0
	LRI	0.1	182	8
Envt B	TSE	0.1	557	0
	LRI	0.01	4402	0
	LRI	0.04	620	8

Reward probabilities are:

Envt A	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
Envt B	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

CHAPTER 4

DISCRETE ESTIMATOR ALGORITHMS

4.1 Introduction

The purpose of this chapter is to combine the approaches of the previous two chapters 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 and the TS Estimator algorithm will be considered in this chapter. Both algorithms are ϵ -optimal; a proof of this will be given. The interesting feature of this proof is that a Discrete Estimator algorithm only needs two properties for it to be ϵ -optimal. First these properties will be stated, and then the algorithms will be introduced.

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. This, of course, is a reasonable constraint because if the action probability goes to zero too quickly, this action will not be sampled an adequate number of times so as to guarantee an accurate estimate of the reward probability. 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 Discrete Estimator algorithm with r actions and a resolution parameter n is said to possess the property of moderation if the magnitude by which an action probability can decrease per iteration is bounded by $1/rn$.

The second property is as intuitively obvious as the first. If the estimate of reward of an action, say α_m , remains the maximum estimate subsequent to a certain point of time, then it would be desirable to have the DEA steadily increase the probability of choosing α_m to unity. Again this seems quite logical. The basis of estimator algorithms is to use estimates of reward probabilities to decide which action to converge to. It only makes sense that it converges to the one with the highest estimate of reward !

Property 2: Suppose there exists an index m and a time instant $t_0 < \infty$, such that $d'_m(t) > d'_j(t)$ for all j such that $j \neq m$ and all $t \geq t_0$. A DEA is said to possess the monotone property if then there exists an integer n_0 such that for all resolution parameters $n > n_0$, $p_m(t) \rightarrow 1$ with probability one as $t \rightarrow \infty$.

We shall first present the Discrete Pursuit Algorithm. It will be shown that this algorithm has both the moderation and the monotone properties. Next the discrete TS Estimator Algorithm will be presented. This algorithm also possesses both of the above properties. Finally a general proof will be given that states that any estimator algorithm with both of the above properties is ϵ -optimal.

4.2 The Discrete Pursuit Algorithm

The primary goal of the Discrete Pursuit Automaton (DPA) is to mimic the learning process followed by the continuous Pursuit Algorithm,

except that the probability changes are made in discrete steps. When the LRI scheme was made discrete, multiplication by λ was replaced by the addition and subtraction of $1/n$. This same philosophy is used for the discrete pursuit algorithm. The only difference is that instead of increasing the probability of choosing the action that has just been rewarded, the probability is increased for the action that has the current maximum estimate of the reward probabilities.

Thus the algorithm works in three steps just like its continuous counterpart, 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.

Note that the above algorithm can be implemented using primarily integer counter updates. This is because the action probability for any action α_i need not be stored as a real number, but only as an integer k_i , whence the action probability can be computed as $k_i \Delta$. Explicitly, the algorithm is as described below.

ALGORITHM Discrete Pursuit Algorithm

Parameters m : the index of the largest component of the estimate vector satisfies $d'_m(t) = \text{Max}_i \{ d'_i(t) \}$, and
 Δ = $1/rn$ is the smallest step size.

$W_i(t)$: an integer which is the number of times the i th action has been rewarded up to time t .

$Z_i(t)$: an integer which is the number of times the i th action has been chosen up to time t .

Initialization $p_i(0) = 1/r$ for all $1 \leq i \leq r$.

Initialize $\mathbf{D}'(t)$ by picking each action a small number of times

Repeat:

Step 1. At time t pick $\alpha(t)$ according to probability distribution $\mathbf{P}(t)$.

Step 2: Let $\alpha(t) = \alpha_k$, update $\mathbf{P}(t)$ based on the components of $\mathbf{D}'(t)$.

if $\beta(t) = 0$ and $d'_m(t) \neq 1$

$$p_j(t+1) = \mathbf{Max}\{ p_j(t) - \Delta, 0 \} \quad \text{for all } j \neq k$$

$$p_k(t+1) = 1 - \sum_{j \neq k} p_j(t+1)$$

otherwise

$$p_j(t+1) = p_j(t) \quad \text{for all } 1 \leq j \leq r \quad (4.1)$$

Step 3: Update $\mathbf{D}'(t)$ according to the following:

$$\text{if } \alpha(t) = \alpha_k, \quad W_k(t+1) = W_k(t) + (1 - \beta(t))$$

$$Z_k(t+1) = Z_k(t) + 1$$

$$d'_k(t+1) = W_k(t+1) / Z_k(t+1)$$

$$\text{for all } i \neq k \quad W_i(t+1) = W_i(t)$$

$$Z_i(t+1) = Z_i(t)$$

$$d'_i(t+1) = d'_i(t) ..$$

End Repeat

END ALGORITHM Discrete Pursuit Algorithm

With the formal presentation of the algorithm complete, the fact that this algorithm has the two necessary properties will be demonstrated.

Lemma 4.1 The DPA possess the moderation property.

Proof: It must be shown that for the DPA the magnitude that any action probability can decrease at any iteration is bounded by $1/r_n$.

The worst case (in fact, the only case) when a component of $P(t)$ can decrease, is when it is decremented by Δ as in (4.1). The result follows since $\Delta = 1/r_n$. ●●●

The second property takes a little more work to prove. It is stated as follows.

Lemma 4.2 The DPA possess the monotone property.

Proof : Suppose for the DPA, there exists an index m and a time instant $t_0 < \infty$ such that $d'_m(t) > d'_j(t)$ for all j such that $j \neq m$ and all $t \geq t_0$, then we have to prove that there exists an integer n_0 , such that for all resolution parameters $n > n_0$, $p_m(t) \rightarrow 1$ with probability one as $t \rightarrow \infty$.

Consider the submartingale defined by the sequence of random variables $\{p_m(t)\}_{t \geq 0}$ satisfying $\sup_{t \geq 0} E[|p_m(t)|] < \infty$. It is known [9] that under these conditions there exists a random variable to which $\{p_m(t)\}$ converges with probability one, i.e.

$$\Pr\left\{\lim_{t \rightarrow \infty} p_m(t) = p_m^\infty\right\} = 1$$

Using the Discrete Pursuit algorithm, we know that if m satisfies

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

then, $d'_m(t) > d'_j(t)$ for all $j \neq m$ and all $t \geq t_0$. Therefore, for all $t > t_0$,

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

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 $j \neq m$. 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 $d'_m(t) > d'_j(t)$. Assuming the algorithm has not converged, there exists at least one non-zero component of $\mathbf{P}(t)$, say $p_k(t)$, and hence we assert that

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

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

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

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

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

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

$$\begin{aligned} \mathbf{E}[p_m(t+1) \mid \mathbf{P}(t), \mathbf{D}'(t), p_m(t) \neq 1] &= d_m \{ p_m(t) + \chi\Delta \} + (1 - d_m) \{ p_m(t) \} \\ &= p_m(t) + d_m \chi \Delta \end{aligned}$$

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

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

$\sup_{t \geq 0} \mathbf{E}[p_m(t+1) \mid \mathbf{P}(t), \mathbf{D}'(t), p_m(t) \neq 1] < \infty$. Thus,

$$\mathbf{E}[p_m(t+1) - p_m(t) \mid \mathbf{P}(t), \mathbf{D}'(t)] = d_m \chi \Delta \geq 0 \text{ or all } t \geq t_0.$$

implying that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [9], $\{p_m(t)\}$ converges and so as $t \rightarrow \infty$,

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

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

consequently that $\sum_{j \neq m} \text{Max}(p_j(t) - \Delta, 0) \rightarrow 0$ w.p. 1. Hence $p_m(t) \rightarrow 1$ w.p. 1, and the result is proved. ●●●

4.3 The Discrete TS Estimator Algorithm

Like its continuous predecessor, the Discrete TS Estimator algorithm (DSTE) 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 $\Delta = 1/r\theta$. Here θ represent the largest integer multiple of Δ that any one component of the probability vector can decrease by in one iteration. The original continuous updating rule contains three factors:

- 1) λ
- 2) $f(d'_i(t) - d'_j(t))$
- 3) $(S_{ij}(t) p_i(t) + S_{ji}(t) \frac{(1-p_i(t)) p_i(t)}{r-1})$

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 $f(d'_i(t) - d'_j(t))$ 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 $(1-p_j(t)) p_i(t)$ 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 $(S_{ij}(t) + S_{ji}(t) \frac{1}{r-1})$. This term is not essential for ϵ -optimal convergence of the scheme. However we have chosen to retain it to preserve the fact that if a component is increased because another action is rewarded, it should not be weighted as much as it would have been if a component is increased when *its* corresponding action is rewarded. The way these terms are brought together is:

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

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 of reward than d'_i . In order 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

$$\text{Rnd} : \mathfrak{R} \rightarrow \mathbb{Z}.$$

This function rounds up its parameter so that its value is always an integer.

The second function is defined as

$$\text{Check} : [0, 1] \times [0, 1] \times \{-\theta, -\theta+1, \dots, \theta-1, \theta\} \rightarrow \{-\theta, -\theta+1, \dots, \theta-1, \theta\}.$$

Given three parameters, p_i , p_j , and x , $\text{Check}(p_i, p_j, x)$ 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. As a final note, 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.

Using these functions, this algorithm is formally described below.

ALGORITHM Discrete TS Estimator

Parameters

$$\Delta = 1/m\theta$$

n : an integer, is a resolution parameter,

r : an integer, is the number of actions,

θ : an integer, is the maximum any one component can change,

$S_{ij}(t)$: is an indicator function such that

$$S_{ij}(t) = 1 \text{ if } d'_i(t) > d'_j(t)$$

$$= 0 \text{ if } d'_i(t) \leq d'_j(t)$$

f : is any monotonic, increasing function, $f:[0, 1] \rightarrow [0, 1]$,
satisfying $f(0) = 0$ and $f(1) = 1$

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

$\text{Check}(p_i(t), p_j(t), x)$ returns the largest integer value of x that
preserves $0 \leq p_i(t) + x\Delta, p_j(t) - x\Delta \leq 1$

Initialization $p_i(0) = 1/r$ for all $1 \leq i \leq r$.

Initialize $\mathbf{D}'(t)$ by picking each action a small number of times.

Repeat:

Step 1: Same as DPA.

Step 2: Let $\alpha(t) = \alpha_j$, update $P(t)$ according to the following:

for the i^{th} component,

FOR each action i , starting with m **DO**

$$\text{change} = \text{Rnd}\left(\theta f(d'_i(t) - d'_j(t)) \left(S_{ij}(t) + S_{ji}(t) \frac{1}{r-1} \right)\right)$$

$$p_j(t+1) = p_j(t) - \Delta \text{Check}(p_i(t), p_j(t), \text{change})$$

$$p_i(t+1) = p_i(t) + \Delta \text{Check}(p_i(t), p_j(t), \text{change}) \quad (4.2)$$

END FOR ... DO

Step 3: Same as DPA.

End Repeat

END ALGORITHM Discrete TS Estimator

Lemma 4.3 The DTSEA 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 $1/rn$.

There are two possible worst cases:

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

$$\Delta \text{Rnd}\left(f(1) \theta\right) = \frac{1}{m\theta} \text{Rnd}\left(\theta\right) = \frac{1}{m}.$$

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

$$\Delta \text{Rnd}\left(f(1) \frac{\theta}{r-1}\right) = \frac{r-1}{m\theta} \text{Rnd}\left(\frac{\theta}{r-1}\right) = \frac{1}{m}.$$

Hence the result. ●●●

Lemma 4.4 The DTSEA processes the monotone property.

Proof : Suppose there exists an index m and a time instant $t_0 < \infty$ such that $d'_m(t) > d'_j(t)$ for all $j \neq m$ and all $t \geq t_0$. Then we need to show that there exists an integer n_0 such that for all resolution parameters $n > n_0$, $p_m(t) \rightarrow 1$ with probability one as $t \rightarrow \infty$.

Again consider the submartingale defined by the sequence of random variables $\{p_m(t)\}_{t \geq 0}$ satisfying $\sup_{t \geq 0} \mathbf{E}[|p_m(t)|] < \infty$. It is known [9], that under these conditions there exists a random variable to which $\{p_m(t)\}$ converges with probability one, i.e.

$$\Pr\left\{\lim_{t \rightarrow \infty} p_m(t) = p_m^\infty\right\} = 1$$

Using the DTSE algorithm, we know that if m satisfies

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

then, $d'_m(t) > d'_j(t)$ for all $j \neq m$ and all $t \geq t_0$. Therefore, for all $t > t_0$,

$S_{mj}(t) = 0$, and so

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

$$= 1 - \Delta \sum_{j \neq m} \text{Rnd}\left(f(d'_m(t) - d'_j(t)) \theta\right)$$

$$\text{if } \beta(t) = 0 \quad \text{(that is w.p. } d_m)$$

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 $j \neq m$. 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 $d'_m(t) > d'_j(t)$. Assuming 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

$$\begin{aligned} p_k(t+1) &= p_k(t) - \frac{1}{m\theta} \text{Rnd}\left(f(d'_m(t)-d'_k(t))\theta\right) \\ &< p_k(t) - \Delta. \end{aligned}$$

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

$$\begin{aligned} E[p_m(t+1) \mid \mathbf{P}(t), \mathbf{D}'(t), p_m(t) \neq 1] &> d_m \{p_m(t) + \Delta\} + (1-d_m) \{p_m(t)\} \\ &= p_m(t) + d_m \Delta \end{aligned}$$

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

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

Hence $\sup_{t \geq 0} E[p_m(t+1) \mid \mathbf{P}(t), \mathbf{D}'(t), p_m(t) \neq 1] < \infty$.

Thus, $E[p_m(t+1) - p_m(t) \mid \mathbf{P}(t), \mathbf{D}'(t)] = d_m \Delta \geq 0$ or all $t \geq t_0$,

implying that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [9], $\{p_m(t)\}$ converges and so as $t \rightarrow \infty$,

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

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 $p_m(t) \rightarrow 1$ w.p. 1, and the lemma is proved. ●●●

With this done, the next step is to investigate the convergence of any scheme possessing the moderation and monotone properties.

4.4 Proof Of Convergence

The asymptotic property of the discrete estimator algorithms will be proved. The proofs of Lemmas 4.2 and 4.4 are analogous to that of the TS

Estimator Algorithm [42] 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 [42], this proof is achieved without the erroneous assumption that the random vectors $\mathbf{P}(t)$ and $\mathbf{P}(t+1)$ are independent.¹⁴ With these modifications accounted for, Theorem 4.2 can be proved using the same method as its continuous counterpart [42].

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 $\epsilon > 0$ and $\delta > 0$, there exist an $n_0 > 0$ and a $t_0 < \infty$ such that for all time $t \geq t_0$ and for any resolution parameter $n > n_0$ the following is true:

$$\Pr[|1 - p_b(t)| < \epsilon] > 1 - \delta.$$

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

¹⁴ We are grateful to Prof. John Dixon, Dept. of Mathematics, Carleton University, for pointing out that, in general, this assumption is false.

up so that each action can be sampled a certain number of times by a suitably choosing the internal parameter n , and thus obtaining estimates of the reward probabilities which are accurate enough so as to yield an accurate discrimination between the actions. This fact is not obvious, and so we will first prove that any DEA has this property.

Theorem 4.1

For each action α_i , assume $p_i(0) \neq 0$. Then for any given constants $\delta > 0$ and $M < \infty$, there exist $n_0 < \infty$ and $t_0 < \infty$ such that under the DEA, for all learning parameters $n > n_0$ and all time $t > t_0$:

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

Proof :

Define the random variable Y_i^t as the number of times the i^{th} action was chosen up to time t . Then we must prove that $\Pr\{Y_i^t > M\} \geq 1 - \delta$. (4.3)

$$\text{Clearly (4.3) is equivalent to } \Pr\{Y_i^t \leq M\} \leq \delta. \quad (4.4)$$

$$\text{The events } \{Y_i^t = k\}, \{Y_i^t = j\} \text{ are mutually exclusive for } j \neq k, \text{ and so} \\ (4.4) \text{ simplifies to } \sum_{k=1}^M \Pr\{Y_i^t = k\} \leq \delta. \quad (4.5)$$

For any iteration of the algorithm, $\Pr\{\alpha_i \text{ is chosen}\} \leq 1$. As well, by the moderation property, during any of the first t iterations of the algorithm: $\Pr\{\alpha_i \text{ is not chosen}\} \leq (1 - p_i(0))^{t/n}$.

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.

$$\Pr\{Y_t \leq M\} < \sum_{k=1}^M C(t,k) (1)^k (1 - p_i(0) + t/m)^{t-k} \quad (4.6)$$

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

As well, $C(t,m) \leq t^m$ which leads to the following:

$$M t^m (1 - p_i(0) + t/m)^{t-m} \leq \delta.$$

Now in order to get the R.H.S. of this term less than δ as t increases, the $(1 - p_i(0) + t/m)$ 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 $(1 - p_i(0) + t/m) < 1$. We do this by requiring that $n > \frac{t}{r p_i(0)}$. Let,

$$n = \frac{2t}{r p_i(0)} \quad (4.7)$$

With this value of n (4.6) simplifies to $\Pr\{Y_t \leq M\} < M t^m \psi^{t-m}$, where $\psi = 1 - 1/2 p_i(0)$ and $0 < \psi < 1$.

It remains to evaluate, $\lim_{t \rightarrow \infty} M t^m \psi^{t-m}$. This is equivalent to,

$$M \lim_{t \rightarrow \infty} \frac{t^m}{(1/\psi)^{t-m}} \quad \text{with } n = \frac{2t}{r p_i(0)} \quad (4.8)$$

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

$$M \lim_{t \rightarrow \infty} \frac{m!}{(\ln 1/\psi)^m (1/\psi)^{t-m}} = 0, \quad \text{with } n = \frac{2t}{r p_i(0)}$$

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 than 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: $\Pr \{ (Y_{t_{(i)}} \leq M) \text{ and (any other event)} \} \leq \Pr \{ (Y_{t_{(i)}} \leq M) \}$.

Thus in this case too, the inequality (4.6) is preserved. Thus for any action α_i : $\Pr \{ (Y_{t_{(i)}} \leq M) \} \leq \delta$ whenever $t > t_{(i)}$ and $n > n_{(i)}$.

Since, we can repeat this argument for all the actions, we can define t_0 and n_0 as follows: $t_0 = \text{Max}_{1 \leq i \leq r} \{ t_{(i)} \}$ and $n_0 = \text{Max}_{1 \leq i \leq r} \{ n_{(i)} \}$. Thus for all i we have shown that for all $t > t_0$ and all $n > n_0$, the quantity $\Pr \{ Y_t < M \} < \delta$, and the theorem is proved. $\dots$

With this property established we are now in a position to prove the central result of this thesis.

Theorem 4.2

In every stationary random environment, the DEA is ϵ -optimal.

Proof : We need to show that given $\epsilon > 0$ and $\delta > 0$, there exists a $n_0 > 0$ and a $t_0 < \infty$ such that for all $t \geq t_0$ and $n > n_0$:

$$\Pr[|p_m(t) - 1| < \epsilon] > 1 - \delta$$

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 $d_b - h \geq d_i$ for all $i \neq m$. 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 $\epsilon > 0$,

$$\Pr\left\{ \left| \frac{X_1 + X_2 + \dots + X_t}{t} - \mu \right| > \epsilon \right\} \rightarrow 0 \text{ as } t \rightarrow \infty.$$

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 $M_i < \infty$, such that if α_i is chosen and least M_i times:

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

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

$$\Pr\left\{ \left| d'_m(t) - d_j \right| < h/2 \right\} > 1 - \delta.$$

By Theorem 1 we know that we can define t_0 and n_0 such that for all i , all $t > t_0$ and all $n > n_0$

$$\Pr\left\{ Y_t^i > M \right\} \geq 1 - \delta. \quad (4.9)$$

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 . Hence we have,

$$\begin{aligned} d_m - h/2 &> d_m - h && \text{because } h > 0 \\ &\geq d_i && \text{for all } i \neq m. \end{aligned}$$

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

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

2

OF/DE

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

In Tables 19 and 20, the difference in reward probabilities is decreased by a factor of four between the subsequent environments reported in the tables. Both TS Estimator 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 is increasing by a constant factor as the difference between the penalty

Table 19--The Number of Iterations and Error Rate
for a Fixed Parameter for the TS Estimator Algorithms

<u>Probability of Reward</u>		<u>Continuous</u>		<u>Discrete</u>	
<u>Action 1</u>	<u>Action 2</u>	<u>Iterations</u>	<u>Errors</u>	<u>Iterations</u>	<u>Errors</u>
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 20--The Number of Iterations and Error Rate
for a Fixed Parameter for the Pursuit Algorithms

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

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

CHAPTER 5

SIMULATION RESULTS

5.1 Rate of Convergence and Accuracy

The problem with performing simulation results can be typified with the following trivial automaton: always pick the first action with probability unity. This algorithm is not expedient, nor is it ϵ -optimal in all random environments. However, any time the optimal action is the first one, the trivial automaton converges correctly in a single iteration. The key point here is that no matter how good an algorithm is, there will always be a special case where another algorithm converges quicker and with a better accuracy. The approach taken to measure the performance of automata in this thesis taken here is the same as it is for many computer algorithms. The discrete estimator algorithms will be compared to their continuous counterparts in environments which have been used as benchmarks, namely, the ones mentioned in chapters 2 and 3.

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. The reason is obvious. The way to increase accuracy is to take a larger sample size, and this slows down the rate of convergence of the algorithm. Unfortunately, this has a performance impact; if the automaton keeps selecting a non-optimal action

for the purposes of displaying an asymptotically increased accuracy, it has a higher probability of being penalized while in the transient stage, that is, before it has "learnt" the optimal action.

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 convergence 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 0.9%. 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 15 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 LRI scheme. 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 TS Estimator algorithm, the range is from 4 to 50 % slower than its discrete counterpart. For example, in the environment referred to as Env't A, the DTSEA takes 207 iterations to reach the end state and the TSEA 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.

Table 15--Comparison of the
Discrete and Continuous Estimator Algorithms in 10 Action Environments

Environment	Algorithm	Continuous	Discrete
<u>Env't A</u>	Pursuit	1140	799
	TS Estimator	310	207
<u>Env't B</u>	Pursuit	2570	1770
	TS Estimator	583	563

Reward probabilities are:

Env't A	0.7	0.5	0.3	0.2	0.4	0.5	0.4	0.3	0.5	0.2
Env't B	0.1	0.45	0.84	0.76	0.2	0.4	0.6	0.7	0.5	0.3

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 LRI scheme by Oommen [39]. 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

Table 16--The Number of Iterations Until Convergence
in Two Action Environments for the Pursuit Algorithm

<u>Probability of Reward</u>		<u>Mean Iterations</u>	
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 17--The Number of Iterations Until Convergence
in Two Action Environments for the TS Estimator Algorithm

<u>Probability of Reward</u>		<u>Mean Iterations</u>	
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.

done by Thathachar and Sastry [43]. These extra 20 iterations were then included in the total number of iterations. The ensemble average results are shown in Table 16 and 17.

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 takes an average of 28.8 iterations and the DTSE takes 24 iterations. However, twenty of these iterations are due to the initialization process. So after the estimates of reward are initialized, the TSE takes 8.8 iterations to converge and the DTSE 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 takes 8,500 iterations for to converge, the DTSE requires only 5,600. The advantage of discretizing is obvious.

5.2 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 18.

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 18 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, the discrete algorithms take from 50 to 75 percent of the time that their continuous counterparts take.

Table 18

CPU Time Used per Iteration in a Ten Action Environment		
<u>Algorithm</u>	<u>Continuous</u>	<u>Discrete</u>
Pursuit	2.9 msec	2.3 msec
TS Estimator	14 msec	7.2 msec

5.3 Adaptive Features

One final set of simulations was run to illustrate a desirable property that the TS Estimator algorithms have, 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 19 and 20, the difference in reward probabilities is decreased by a factor of four between the subsequent environments reported in the tables. Both TS Estimator 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 is increasing by a constant factor as the difference between the penalty

Table 19--The Number of Iterations and Error Rate
for a Fixed Parameter for the TS Estimator Algorithms

<u>Probability of Reward</u>		<u>Continuous</u>		<u>Discrete</u>	
<u>Action 1</u>	<u>Action 2</u>	<u>Iterations</u>	<u>Errors</u>	<u>Iterations</u>	<u>Errors</u>
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 20--The Number of Iterations and Error Rate
for a Fixed Parameter for the Pursuit Algorithms

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

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.

CHAPTER 6

CONCLUSION

6.1 Present Role of Discrete Estimator Algorithms

As software is becoming increasingly complex there is a desire for general methods to 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 routine may deal with the storage of customer files for long distance billing. Rather than 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. The sample applications described in Chapter 1 illustrate how learning automata have already been developed to handle all of these problems. Automata are adaptable, and as such represent a fault tolerant approach.

The rate at which learning algorithms converge has been a limiting factor in their implementation in the past. Discretizing provides a general method of improving the performance of VSSA. Discretizing VSSA restricts the probability of choosing an action to a finite number of values. It also reduces the required computation time by replacing floating-point multiplication operations with integer addition operations. Experimentally discretizing also reduces the number of iterations needed for an algorithm to converge while it simultaneously eliminates the need for precise random number generators.

Estimator Algorithms are among the quickest stochastic learning automata known to date. Hence, they represent a natural candidate for discretization. Fortunately, ϵ -optimality is preserved when they are discretized. In fact, in some environments the discrete versions require 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 reduced by a factor ranging between 50 % and 75 %. Thus, as such, Discrete Estimator Algorithms are a useful tool in software design.

6.2 Future Prospects of Discrete Estimator Algorithms

We believe that the Discretized Estimator Algorithms introduced and studied in this thesis are among the quickest converging ϵ -optimal learning algorithms presently known. With regard to open problems, the results of this thesis make Discrete Estimator Algorithms a useful component for the building of networks and hierarchies of learning automata. Also as always, the full range of applications of learning automata remain to be investigated.

REFERENCES

- [1] Baba, S., Soeda, S.T., and Sawaragi, Y., "An Application of Stochastic Automata to the Investment Game," *Int. J. Syst. Sci.*, vol. 11, no. 12, pp 1447-1457, Dec. 1980.
- [2] Booth, Taylor L. *Sequential Machines and Automata Theory*. New York: John Wiley and Sons, Inc., 1967.
- [3] Chachra, V., Ghare, P. M., Moore, M., *Application of Graph Theory Algorithms*. North Holland, 1979.
- [4] Chernicoff, Stephen, *Macintosh Revealed Vol. 1*. Hayden Books, 1987.
- [5] Enderton, Herbert B. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.
- [6] Flerov, Y.A., "Some Classes of Multi-Input Automata", *Journal of Cybernetics*, Vol. 2, 1972, pp.112-122.
- [7] Isaacson, D.L., and Madson, R.W., *Markov Chains : Theory and Applications*, Wiley, 1976.
- [8] James, Robert C.. James, Glenn, *Mathematical Dictionary* 4th, ed., Van Nosrand Reinhold Company, 1976.
- [9] Karlin, S., Taylor, H. M., *A First Course on Stochastic Processes*, 2d ed., Academic Press, 1974.
- [10] Knuth, D. E., *The Art of Computer Programming Vol. 1*. Addison Wesley, 1968.
- [11] Lakshmivarahan, S., *Learning Algorithms Theory and Applications*, Springer-Verlag, 1981.

- [12] Lakshmivarahan, S., " ϵ -Optimal Learning Algorithms - Non-Absorbing Barrier Type", *Technical Report EECS 7901*, February 1979, School of Electrical Engineering and Computing Sciences, University of Oklahoma, Norman, Oklahoma.
- [13] Lakshmivarahan, S., "Two Person Decentralized Team With Incomplete Information", *Applied Mathematics and Computation*, Vol. 8, pp.51-78, 1981.
- [14] 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.
- [15] 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.
- [16] Meybodi, M.R., Lakshmivarahan, S., "A Learning Approach to Priority Assignment in a Two Class M/M/1 Queueing System with Unknown Parameters", *Proc. Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, 1983.
- [17] Motorola, *MC68000 16/32-Bit Microprocessor Programmer's Reference Manual*, 4th ed., Prentice-Hall, 1984.
- [18] Motorola, *MC68020 32-Bit Microprocessor*, 2d ed., Prentice-Hall, 1985.
- [19] Mukhopadhyay, S., Thathachar, M.A.L., "Associative Learning of Boolean Functions", *IEEE Systems Man and Cybernetics Transaction*. To appear Sept/Oct 89.
- [20] Munkres, James R., *Topology a First Course*. Prentice-Hall, 1975.
- [21] Narendra, K.S., and Thathachar, M.A.L., *Learning Automata*, Prentice-Hall, 1989.

- [22] 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.
- [23] 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.
- [24] 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.
- [25] Narendra, K.S., and Lakshmivarahan, S., "Learning Automata : A Critique", *Journal of Cybernetics and Information Sciences*, Vol. 1, 1987, pp.53-66.
- [26] Naur, P. *Concise Survey of Computer Methods*. Petrcelli Books, 1974.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] Oommen, B. J., "Ergodic Learning Automata Capable of Incorporating A priori Information", *IEEE Transactions on Systems, Man and Cybernetics.*, Vol. SMC-17, July/August 1987, pp.717-723.

- [31] 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.
- [32] Oommen, B.J., "A Learning Automaton Solution to the Stochastic Minimum Spanning Circle Problem", *IEEE Trans. on Syst. Man and Cybern.*, July/Aug. 1986, pp.598-603.
- [33] 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.
- [34] 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.
- [35] Pinkava, Václav. *Introduction to Logic for Systems Modelling*. Abacus Press, 1988.
- [36] Ramesh, R., *Learning Automata in Pattern Classification*. M.E. Thesis, Indian Institute of Science, Bangalore, India, 1983.
- [37] Ross, Sheldon M., *Introduction to Probability Models*. 3 rd Ed. San Diego: Academic Press, 1985.
- [38] 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.
- [39] Thathachar, M.A.L., and Oommen, B.J., *Discretized Reward-Inaction Learning Automata*, *Journal of Cybernetics and Information Sciences*, Spring 1979, pp. 24-29.
- [40] 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.
- [41] 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.

- [42] 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.
- [43] Thathachar, M.A.L., and Sastry, P.S., "Pursuit Algorithm for Learning Automata", Transcript in the hand of Dr. B. Jon Oommen, Carleton Univeristy.
- [44] Tsetlin, M.L., "On the Behaviour of Finite Automata in Random Media", *Automat. Telemekh.(USSR)*, Vol.22, Oct. 1961, pp.1345-1354.
- [45] Tsetlin, M.L., *Automaton Theory and the Modelling of Biological Systems*, New York and London, Academic, 1973.
- [46] Tsypkin, Y.Z. and Poznyak, A.S., "Finite Learning Automata", *Engineering Cybernetics*, Vol.10, 1972, pp.478-490.
- [47] Varshavskii, V.I., and Vorontsova, I.P., "On the Behaviour of Stochastic Automata With Variable Structure", *Automat. Telemek.(USSR)*, Vol.24, 1963, pp.327-333.

END

2010 2/9 1

FIN