INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

FAMILIES OF ESTIMATOR-BASED STOCHASTIC LEARNING ALGORITHMS

by

MARIANA AGACHE, B.Sc.

A thesis submitted to the

Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

January 2000

© Copyright

2000, Mariana Agache

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

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

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

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

0-612-52379-9

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Families of Estimator-based Stochastic Learning Algorithms

submitted by **Mariana Agache, B.Sc.**
in partial fulfillment of the requirements for
the degree of Master of Computer Science

Director, School of Computer Science

Thesis Supervisor

CARLETON UNIVERSITY

JANUARY 2000

ABSTRACT

This thesis studies the field of Estimator-based Learning Automata. We first argue that the Reward-Penalty and Reward-Inaction learning paradigms in conjunction with the continuous and discrete models of computation lead to four versions of Pursuit Learning Automata. Such schemes permit the Learning Automaton to utilize the long-term and short-term perspectives of the environment. We present all four resultant Pursuit algorithms, and a quantitative comparison of their performance.

The existing Pursuit Algorithms '*pursue*' the action that is currently estimated as the best action. In this thesis, we claim that by pursuing all the actions with higher estimates than the chosen action, the performance of the Pursuit Algorithms improves considerably. To attest this, we introduce two new Pursuit algorithms, and justify their superiority through extensive simulations.

Thathachar and Sastry introduced the TSE estimator algorithm and presented its updating equations in a scalar form. In this thesis, we present a vectorial representation of the TSE algorithm, and propose its generalized vectorial version, whose superiority is also proven experimentally.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my Professor, Dr. John Oommen, for his continuous support, encouragement, and for believing in me. Throughout these years, his guidance, ideas, and enthusiasm gave me the opportunity to explore new scientific domains and to discover new research horizons, for which I am deeply grateful.

I would also like to thank Professor M. A. L. Thathachar, of the Indian Institute of Science, Bangalore, India, for his valuable comments and suggestions related to the work presented in the Chapter 3 of this thesis.

I would like to thank my parents, my sister and Wellington for their help, support, and encouragement.

DEDICATION

To my parents.

TABLE OF CONTENTS

ABSTRACT	III
ACKNOWLEDGMENTS	IV
DEDICATION.....	V
LIST OF TABLES	X
LIST OF FIGURES.....	XII
CHAPTER 1: INTRODUCTION	1
1.1 General description of the learning automata	1
1.2 Justification and objectives of the Thesis.....	6
1.3 Contributions of the Thesis	7
1.4 Content and organization of the Thesis.....	9
CHAPTER 2: LEARNING AUTOMATA-AN OVERVIEW	11
2.1 Definition of Automaton	11
2.1.1 Deterministic Automaton	12
2.1.2 Stochastic Automaton	13

2.2 The Learning Automaton	15
2.2.1 The Environment.....	16
2.2.2 Definition of the Learning Automaton.....	17
2.2.3 Norms of behavior.....	18
2.3 Fixed Structure and Variable Structure Learning Automata.....	21
2.3.1 Fixed Structure Automata	21
2.3.1.1 Tsetlin Automaton.....	21
2.3.1.2 Krinsky Automaton	24
2.3.1.3 Krylov Automaton.....	26
2.3.2 Variable Structure Stochastic Automata	27
2.3.2.1 Linear Reward-Penalty Scheme (L_{RP}).....	31
2.3.2.2 Linear Reward-Inaction Scheme (L_{RI})	32
2.3.2.3 Linear Inaction-Penalty Scheme (L_{IP}).....	33
2.3.3 Discretized Learning Automata.....	34
2.3.3.1 Discretized Linear Reward-Inaction Automaton	35
2.3.3.2 Discretized Linear Inaction-Penalty Automaton.....	38
2.3.3.3 Discretized Linear Reward-Penalty Automaton.....	39
2.4 Estimator Algorithms	43
2.4.1 Overview	43
2.4.2 Continuous Estimator Algorithms.....	45
2.4.2.1 Pursuit Algorithm.....	45
2.4.2.2 TSE Algorithm	47

2.4.3 Discrete Estimator Algorithms.....	51
2.4.3.1 Discrete Pursuit Algorithm.....	52
2.4.3.2 Discrete TSE Algorithm.....	54
2.5 Conclusions	58
CHAPTER 3: NEW PURSUIT ALGORITHMS.....	60
3.1 Introduction	60
3.2 Continuous Reward-Inaction Pursuit Algorithm (CP_{RI})	62
3.3 Discretized reward-penalty Pursuit Algorithm (DP_{RP})	66
3.4 Simulation Results	69
3.5 Conclusions	75
CHAPTER 4: GENERALIZATION OF THE PURSUIT ALGORITHM	77
4.1 Introduction	77
4.2 Generalized Pursuit Algorithm.....	80
4.3 Discretized Generalized Pursuit Algorithm	86
4.4 Simulation Results	90
4.5 Conclusions	93
CHAPTER 5: GENERALIZATION OF THE TSE ALGORITHM.....	94
5.1 Vectorial representation of the TSE algorithm	94
5.2 Generalization of the TSE algorithm	98
5.3 Simulation Results	105
5.4 Conclusions	109

CHAPTER 6: CONCLUSIONS.....	111
6.1 Summary	111
6.2 Future work	115
REFERENCES	116

LIST OF TABLES

Table 2.1: Experimental comparative performance of DL_{RI} with other FSSA [14].....	38
Table 2.2: Comparative performance of DL_{RI} , ADL_{IP} , ADL_{RP} ($c_2=0.8$).....	41
Table 2.3: Comparison between continuous and discrete linear VSSA.....	42
Table 2.4: The number of iterations until convergence in two-action environments for the TSE Algorithm [10]	56
Table 2.5: Comparison of the discrete and continuous estimator algorithms in benchmark ten-action environments [10].	57
Table 3.1: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$)..	71
Table 3.2: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 500 experiments ($NE=500$)..	71
Table 3.3: Comparison of the Pursuit algorithms in new two-action environments for which exact convergence was required in 750 experiments ($NE=750$).....	73
Table 3.4: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$)..	74
Table 3.5: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 500 experiments ($NE=500$)..	74

Table 4.1: Performance of the generalized Pursuit algorithms in benchmark ten-action environments for which exact convergence was required in 750 experiments (NE=750)	91
Table 5.1: Performance of the GTSE and TSE algorithms in ten-action benchmark environments for which exact convergence was required in 750 experiments (NE=750)	106
Table 5.2: Performance of the GTSE and DTSE algorithms in benchmark ten-action environments for which exact convergence was required in 750 experiments (NE=750)	109

LIST OF FIGURES

Figure 2.1: The automaton.	11
Figure 2.2: The environment.....	16
Figure 2.3: Feedback connection of automaton and environment.	17
Figure 2.4: State transition graphs for the Tsetlin automaton $L_{2N,2}$	22
Figure 2.5: State transition graphs for the Krinsky automaton $K^1_{2N,2}$	25
Figure 2.6: State transition graphs for the Krylov automaton $K^2_{2N,2}$	26
Figure 3.1: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 750 experiments (NE=750).	72
Figure 4.1: Solution approach of the CP_{RP} Pursuit Algorithm and the Generalized Pursuit algorithms	78
Figure 4.2: Performance of the Pursuit Algorithms relative to the CP_{RP} algorithm in ten- action environments for or which exact convergence was required in 750 experiments	92
Figure 5.1: Performance of the continuous Estimator Algorithms relative to the CP_{RP} algorithm in ten-action environments for or which exact convergence was required in 750 experiments.....	107

Figure 6.1: Performance of some Estimator Algorithms relative to the CP_{RP} algorithm	
in ten-action environments for which exact convergence was required in 750	
experiments	114

Chapter 1: INTRODUCTION

1.1 General description of the learning automata

Learning represents one of the most important psychological processes and it is essential in the behavior of self-adjusting organisms. In psychology, it can be defined as an organism's ability to modify its current behavior based on past behavior and the consequences of its prior choices. The psychological and biological concepts of learning have been widely studied and have been incorporated in many engineering systems that deal with incomplete information and uncertainty. Applications such as Adaptive Control Systems, Pattern Recognition, Game Theory, and Object Partitioning, needed to incorporate learning characteristics in their functionality in order to model systems with a substantial amount of uncertainty. Many of these systems are required to choose the correct action (decisions) without *a priori* knowledge of the consequences of performing these actions. To perform well under these conditions of uncertainty, the systems needed to acquire some knowledge about the consequences of performing the various actions. This acquisition and utilization of relevant knowledge in order to improve the performance of a system is posed as the *learning problem*. The goal of the learning problem is to compensate for the insufficient information by appropriate data collection

and processing, while moving the process towards its solution. In these engineering systems, learning has been implemented using various methods and techniques, namely: stochastic approximation methods [3], heuristic programming techniques [23], inductive inferential techniques [25], and statistical inferential techniques [4] [5].

Another approach to solving the learning problem was initiated by the Russian mathematician Tsetlin. He introduced, in 1961, a new model of computer learning that is now called a *learning automaton*, and which will be the focus of the study presented in this thesis. The goal of such an automaton is to determine the optimal action out of a set of allowable actions. The functionality of the learning automaton can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with an *Environment*. The automaton chooses an action that triggers a response from the *Environment*. Such a response can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. The term *Environment* refers, in general, to the collection of all external conditions and influences affecting the life and development of an organism or system. In the context of learning automata, the term defines a random unknown 'media' in which an automaton or a group of automata can operate.

Learning automata can operate individually or can be interconnected in a hierarchical or distributed fashion [30]. Some of the applications using learning automata that operate in such different organizational models are: telephone and traffic routing and control [13], game theory [30], stochastic geometric problems [15], pattern recognition [30], and the stochastic point location problem [20].

Learning automata can be classified with respect to their transition states as being either *deterministic* or *stochastic*. For a deterministic automaton, given an initial state and input, the next state and action are uniquely specified. For a stochastic automaton, given an initial state and input sequence, there is no certainty regarding the next states and actions of the automaton. These automata can further be classified with respect to their transition functions in two categories: *Fixed Structure Stochastic Automata (FSSA)* and *Variable Structure Stochastic Automata (VSSA)*. In the case of FSSA, the state transition and output functions are independent of time, and are thus considered to be of a “fixed structure”. The VSSA are designed with more flexibility, allowing the state transition and output function to vary in time.

The earlier models of stochastic learning automata consider the probability space as a continuous space, the action probabilities being able to take any value in the interval $[0,1]$. In 1979, Thathachar and Oommen [26] opened a new direction in the evolution of the field of learning automata by introducing the concept of *discretized learning automata*. These automata operate in the probability space $[0,1]$, which is divided into a finite number of continuous subintervals, and the probabilities of choosing different actions are allowed to assume values from this finite set. The learning automata designed using this technique exhibit a decrease in their convergence time. They are therefore faster than the learning automata designed to use a continuous probability space.

In the attempt to model the psychological concepts of learning, various VSSA used different learning paradigms. Specifically, the basic concepts of the operant conditioning learning method have been applied in the context of the VSSA algorithms.

These algorithms would update the action probabilities in the following situations: a) when the environment rewarded or penalized an action, b) when the environment rewarded an action, ignoring the penalties, c) when the environment penalized an action, ignoring the rewards.

In the quest to design even faster converging learning algorithms, Thathachar and Sastry [27] introduced a new class of learning automata, called *estimator algorithms*. These algorithms are characterized by the fact that they maintain running estimates for the penalty probability of each possible action, and use them to update the probabilities of choosing each action. Thathachar and Sastry introduced the concepts of the estimator algorithms by first presenting the Pursuit Estimator Algorithm [27]. This learning algorithm ‘pursues’ the action that is considered to have the highest reward estimate. Later, in 1984, the same authors presented another estimator algorithm, the TSE algorithm, which increases the action probabilities for all the actions that have higher reward estimates than the current chosen action. Oommen and Lanctôt continued the study of the estimator algorithms and presented in 1990 discretized versions of the Pursuit and TSE algorithm.

Learning Automata can be used to solve other, more general, learning problems or can be employed as basic learning elements of other learning machines. For instance, the research on LA directly influenced the trial-and-error thread of *reinforcement learning*, leading to modern reinforcement learning research [24]. Due to their learning philosophy, LA can be employed in solving simplified reinforcement learning problems. Specifically, Sutton and Barto in [24] showed how L_{RI} , L_{RP} , and the Pursuit schemes can

be used in solving evaluative feedback problems such as the n-armed bandit problem. Because the LA learn to choose the optimal action from a set of allowable actions, the structure proves to be impractical for solving the *full reinforcement learning problem*, which has as goal to maximize the total amount of reward received over the long run. A method that solves the full reinforcement problem is the *Q-learning* method, whose objective is to find a *control rule* that maximizes at each time step the expected discounted sum of future reward. From this perspective, the LA differ from the *Q-learning* method because they attempt only to learn the optimal action, independently of the expected sum of rewards.

Narendra and Thathachar [11] presented various models on interconnected learning automata, such as synchronous and sequential modes, hierarchies and networks of learning automata. The last model of interconnected automata was strongly influenced by the multilayered artificial neural networks or connectionist networks, but with some underlying differences. Specifically, the networks of learning automata differ from the neural network model in the way the networks are adjusted. The neural networks are adjusted based on the error between the output of the network and some desired output, whereas the adjustment in the network of automata depends on the random response from the environment. In both cases, their performance functions are improved by the adjustment of the weight vectors. Further research is necessary in the field of Learning Automata to determine the weights of a learning network using the learning automata schemes.

1.2 Justification and objectives of the Thesis

This thesis concentrates on the study of the estimator algorithms, focusing primarily on the introduction of new and better-performing estimator algorithms. It also studies the characterization of their performance in comparison to the existing estimator algorithms.

The original Pursuit algorithm presented by Thathachar and Sastry is a continuous algorithm that updates the action probabilities whenever the environment rewards and penalizes an action. Oommen and Lanctôt [18] later extended the Pursuit algorithm into the discretized world by presenting the Discretized Pursuit Algorithm based on the reward-inaction learning paradigm. The Reward-Penalty and Reward-Inaction learning paradigms in conjunction with the continuous and discrete models of computation leads to four versions of Pursuit Learning Automata, but only two of them have been presented in the literature. This represents a gap in the class of the learning automata, which we address. Hence, one of the objectives of this study is to introduce the new versions of Pursuit algorithms that completely covers all possible versions of these algorithms, and to present a thorough comparison of their performance based on simulation results.

The Pursuit algorithm ‘pursues’ the action that is currently estimated as the best action. This implies that, if at a certain time in the evolution of the automaton, the action that is estimated to be the best action is not the action with minimum penalty probability, the automaton pursues a wrong action. This behavior of the automaton can considerably increase its convergence time in such environments and we consider this as a limitation in the design of Pursuit algorithms. Therefore, another objective of this thesis is to improve

the design of the Pursuit algorithm in order to minimize the probability of pursuing a wrong action, and to increase its convergence performance.

Oommen and Lanctôt [18] presented a continuous version of the Pursuit in a vectorial form, allowing for a better understanding of the concepts of the Pursuit algorithm. To better outline the underlying concepts of the TSE algorithm, a goal of our study is to generate a vectorial form of the TSE algorithm, and to propose new directions of improvement of the performance of the TSE algorithm.

In summary, the main objectives of this thesis are:

- to create new Pursuit algorithms that utilize in their design the reward-penalty and reward-inaction learning paradigms, and to study their performance;
- to create new Pursuit algorithms that minimize the probability of choosing a wrong action, therefore exhibiting a better convergence performance;
- to generate a vectorial representation of the updating equations of the TSE algorithm and to determine new directions of improving the performance of the TSE algorithm.

1.3 Contributions of the Thesis

In this thesis, we introduce five new estimator algorithms: the Continuous Pursuit Reward-Inaction Algorithm (CP_{RI}), the Discrete Pursuit Reward-Penalty Algorithm (DP_{RP}), the Generalized Pursuit Algorithm (GPA), the Discretized Generalized Pursuit Algorithm (DGPA), and the Generalized TSE Algorithm (GTSE).

The Continuous Pursuit Reward-Inaction (CP_{RI}) and the Discrete Pursuit Reward-Penalty (DP_{RP}) algorithms, along with the existing Continuous Pursuit Reward-Penalty (CP_{RP}) and Discrete Pursuit Reward-Inaction (DP_{RI}) algorithms, completely cover all the possible versions of Pursuit algorithms that use the Reward-Penalty and Reward-Inaction learning paradigms. The experimental results obtained during this study prove that among these algorithms, the DP_{RI} is the fastest one. Furthermore, we have found that the Reward-Inaction schemes are generally superior to their Reward-Penalty counterparts, and we have experimentally verified that the discretized schemes exhibit a better performance than the continuous schemes.

The search for a new direction of improving the performance of the Pursuit algorithm led to the development of the generalized Pursuit algorithms GPA and DGPA. These algorithms generalize the concepts of the original Pursuit algorithm by pursuing more than one action, therefore minimizing the probability of pursuing a wrong action and increasing the performance of these algorithms. The Generalized Pursuit Algorithm (GPA) proves to be the fastest continuous Pursuit algorithm, and the Discretized Generalized Pursuit Algorithm (DGPA) proves to be the fastest discretized Pursuit algorithm and, more generally, the fastest Pursuit algorithm reported to date.

Another contribution of this thesis is the development of an original vectorial representation of the updating equations of the TSE algorithm. This vectorial representation open a new direction of generalizing the TSE algorithm, and leads to the introduction of the novel GTSE algorithm. This new algorithm experimentally proves to be the fastest continuous estimator algorithm reported to date.

All the novel algorithms proposed in this thesis exhibit very good convergence properties. Also, all of these algorithms have been proven ϵ -optimal in any stationary random environment.

1.4 Content and organization of the Thesis

This thesis debuts with an overview of the field of learning automata in Chapter 2, presenting first the underlying mathematical models, and also describing the measures of the performance of these algorithms, including expediency, optimality, ϵ -optimality and absolute expediency. Also, in this chapter we present the concepts of the Fixed Structure and Variable Structure Stochastic Automata, along with various linear schemes that use different learning updating paradigms such as the reward-penalty, the reward-inaction and the inaction-penalty. Moreover, the continuous and discretized models of learning automata are portrayed through the examples of various such learning schemes.

Chapter 3 addresses the first objective of this thesis, and contains a study of the Pursuit algorithms with respect to different learning paradigms such as reward-penalty and reward-inaction. The novel work presented in this chapter was published in the technical paper [16].

Chapter 4 presents two new generalized Pursuit algorithms, along with a relative comparison of their performance. The work presented in this chapter addresses the second objective of this thesis, which is to improve the performance of the Pursuit estimator algorithms by minimizing the probability of pursuing a wrong action.

The vectorial representation of the updating equation of the TSE algorithm is presented in Chapter 5, together with a new generalized TSE algorithm and an evaluation of its performance.

Chapter 6 concludes the thesis by presenting a summary of the results, and outlining the main conclusions of this work.

Chapter 2: LEARNING AUTOMATA-AN OVERVIEW

2.1 Definition of Automaton

An automaton is defined, in general, as a system that directs itself and does not need exterior guidance in order to function. Mathematically, an automaton is defined as a set of input actions B , a set of states Q , a set of output actions A , and the functions F and G needed to compute the next state of the automaton and, its output respectively. The input determines the evolution of the automaton from its current state to the next state, as in Figure 2.1. If the output depends only on the current state, the automaton is defined as a *state-output automaton*.

Figure 2.1: *The automaton.*

The following is a formal definition of an automaton:

Definition 2.1: An *automaton* is a quintuple $\langle A, B, Q, F, G \rangle$, where:

- $A = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ is the set of output actions of the automaton, $2 \leq r < \infty$.
- B is the set of input actions that can be finite or infinite.
- Q is the vector state of the automaton with $q(t)$ denoting the state at instant t , as:

$$Q = \{q_1(t), q_2(t), \dots, q_s(t)\}$$

- $F: Q \times B \rightarrow Q$ is the *transition function* that determines the state at the instant $t+1$ in terms of the state and input at the instant t :

$$q(t+1) = F(q(t), \beta(t)).$$

This mapping can be either deterministic or stochastic.

- The output function G determines the output of the automaton at any instant ' t ' based on the state at the current instant:

$$\alpha(t) = G(q(t)).$$

The mapping $G: Q \rightarrow A$ can, with no loss of generality, be considered deterministic [11].

The automaton is considered to be *finite* if the sets Q , B and A are all finite.

The automaton is considered deterministic or stochastic as described below.

2.1.1 Deterministic Automaton

The automaton is a *deterministic automaton* if both F and G are deterministic mappings. For such an automaton, given an initial state and input, the next state and action are uniquely specified.

2.1.2 Stochastic Automaton

If F or G is stochastic, the automaton is called a *stochastic automaton*. If the state transition mapping F is stochastic, given an initial state and input sequence, there is no certainty regarding the states and actions that follow. We can only consider the probabilities of reaching various states. For this reason, F can be specified in terms of the conditional probability matrices $F^{\beta^1}, F^{\beta^2}, \dots, F^{\beta^m}$, where each F^{β} for $\beta \in B$ is a $s \times s$ matrix whose entries are given by:

$$\begin{aligned} f_{ij}^{\beta} &= \Pr\{q(t+1) = q_j \mid q(t) = q_i, \beta(t) = \beta\}, & i = 1, \dots, s \\ & & j = 1, \dots, s \\ & & \beta = \beta_1, \beta_2, \dots, \beta_m. \end{aligned} \quad (2.1)$$

Hence, f_{ij}^{β} represents the probability that the automaton moves from state q_i to q_j on receiving the input β . Based on the fact that all f_{ij}^{β} are probabilities, and that after a transformation the automaton has to reach one of its states, we have:

$$\sum_{j=1}^s f_{ij}^{\beta} = 1, \text{ for each } \beta \in B, \quad (2.2)$$

and, consequently, F^{β} is a Markov matrix.

If the mapping G is stochastic, it can be represented by a conditional probability matrix of dimension $s \times r$ having the following elements:

$$g_{ij} = \Pr\{\alpha(t) = \alpha_j \mid q(t) = q_i\}, \quad i = 1, \dots, s, \quad j = 1, \dots, r. \quad (2.3)$$

In the above equation, g_{ij} denotes the probability that the automaton chooses α_j given that it is in state q_i . Since g_{ij} are probabilities, it implies:

$$\sum_{j=1}^r g_{ij} = 1, \text{ for each } i = 1, \dots, s. \quad (2.4)$$

It can be shown [22] that by a proper redefinition of states, the output function G of any stochastic automaton can be made deterministic by increasing the number of states in the automaton.

If the conditional probabilities f_{ij}^{β} and g_{ij} are independent of both t and the input sequence, the stochastic automaton is called a *fixed structure stochastic automaton* (FSSA). If the transition probabilities, f_{ij}^{β} vary based on the input at each step t , the automaton is called a *variable structure stochastic automaton* (VSSA).

If the transition mapping is stochastic, we can not determine precisely the state of the automaton at a given time. Indeed, we can only calculate the probability with which the automaton is in a particular state at a given instant. These probabilities are known as *state probabilities*. The *state probability vector* can be defined as $\boldsymbol{\pi}(t) = [\pi_1(t), \pi_2(t), \dots, \pi_s(t)]^T$, where $[\]^T$ denotes the transposed matrix and

$$\pi_j(t) = \Pr\{q(t) = q_j\}. \quad (2.5)$$

Given the input and the initial state probability vector $\boldsymbol{\pi}(0)$, the state probability vector at $t=1$ is obtained as follows:

$$\begin{aligned} \pi_j(1) &= \Pr\{q(1) = q_j \mid \beta(0)\} \\ &= \sum_{i=1}^s \Pr\{q(1) = q_j \mid q(0) = q_i, \beta(0)\} \cdot \Pr\{q(0) = q_i\} \\ &= \sum_{i=1}^s f_{ij}^{\beta(0)} \pi_i(0). \end{aligned} \quad (2.6)$$

In vector form, the equation can be written as:

$$\pi(1) = [F^{\beta(0)}]^T \cdot \pi(0). \quad (2.7)$$

Recursively, this leads to the state probability vector at time t , as:

$$\pi(t) = [F^{\beta(t-1)}]^T [F^{\beta(t-2)}]^T \dots [F^{\beta(0)}]^T \pi(0). \quad (2.8)$$

Similarly, the components of the action probability vector $P(t)$ are defined as:

$$p_i(t) = \Pr\{\alpha(t) = \alpha_i\} \quad i = 1, \dots, r \quad (2.9)$$

which can be seen to be related to $\pi(t)$ by [11]:

$$P(t) = G^T \cdot \pi(t). \quad (2.10)$$

This equation shows that one can relate the state probability $\pi(t)$ to the action probability $P(t)$, and thus perform the above computations by first processing $\pi(t)$ and then computing $P(t)$ using the output matrix.

2.2 The Learning Automaton

The goal of an automaton is to determine the optimal action from a set of possible actions. The automaton performs these actions in a random environment that generates a response for each action. The following section of this chapter describes the characteristics of the environment in which a learning automaton operates.

2.2.1 The Environment

The environment (see Figure 2.2) can be defined mathematically by a triple $\{A, C, B\}$ where $A = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ represents a finite input set, $B = \{\beta_1, \beta_2, \dots, \beta_l\}$ ($2 \leq l < \infty$) is the output set of the environment and $C = \{c_1, c_2, \dots, c_r\}$ a set of penalty probabilities, where each element c_i of C corresponds to an input action α_i .

Figure 2.2: The environment

At any discrete time t ($t=0,1,2,\dots$) an input $\alpha(t)$ can be applied to the environment which will generate an output $\beta(t)$. Usually, the output set of an environment has two elements β_1 and β_2 , which are considered to be 0 and 1 for mathematical convenience. As a convention, an output $\beta(t)=1$ is considered a *failure* or an *unfavorable response* or a *penalty* and an output $\beta(t)=0$ is considered a *success* or a *favorable response* or a *reward*. The systems that interact with an environment that generates only two output values are considered *P-models*. If the output of the environment is a finite set, the systems interacting with this type of environment are referred to as *Q-models*. As a further generalization, when the output of the environment is a continuous random variable, which assumes values in the interval $[0,1]$, the model is referred to as an *S-model*.

The environments can also be classified based on their evolutionary properties. If the penalty probabilities c_i ($i=1,2,\dots,r$) are constant in time, the environment is called a

stationary environment. However, if one or more penalty probabilities c_i ($i=1,2,\dots,r$) are not constant, the environment is considered *nonstationary*.

2.2.2 Definition of the Learning Automaton

A “learning” automaton is an automaton that interacts with a random environment, having as goal to improve its behavior. It is connected to the environment in a feedback loop, such that the input of the automaton is the output of the environment and the output of the automaton is the input for the environment, as shown in Figure 2.3.

Figure 2.3: Feedback connection of automaton and environment.

Starting from an initial state $q(0)$, the learning automaton generates the corresponding action $\alpha(0)$. The environment generates a response to this action, $\beta(0)$, which dictates to the automaton that based on its transition matrix F , it should change its state to $q(1)$. This cycle is repeated until the probability of choosing the action that has the smallest penalty probability, hopefully, becomes as close to unity as desired.

2.2.3 Norms of behavior

In order to define some quantitative norms of behavior for the learning automata, the automata are considered to operate in a stationary random environment with the penalty probabilities $\{c_1, c_2, \dots, c_r\}$. If two automata operate in such an environment, the automaton that receives a bigger number of favorable responses from the environment is considered better. To achieve this, the automaton has to learn to choose the “best” action, where the “best” action is considered the action with minimum penalty probability $c_{\min}$.

One basic method by which one could learn to choose the best action is based on the pure chance approach. If there is no a priori information regarding each action, it is not possible to distinguish between the different actions. In such case, each action is chosen with equal probability $p_i(t) = 1/r, i=1,2,\dots,r$. An automaton that uses this learning approach is called a “pure-chance automaton” and it is considered a standard for comparison of the behavior of the learning automata.

In order to compare various learning automata, the *average penalty* for a given action probability vector $\mathbf{P}(t)$ at time ‘t’ is defined as:

$$\begin{aligned} M(t) &= E[\beta(t) | \mathbf{P}(t)] = \Pr[\beta(t) = 1 | \mathbf{P}(t)] \\ &= \sum_{i=1}^r \Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \cdot \Pr[\alpha(t) = \alpha_i] \\ &= \sum_{i=1}^r c_i p_i(t) \end{aligned} \quad (2.11)$$

For the “pure-chance” automaton, the average penalty is calculated to be [11]:

$$M_0 = \frac{1}{r} \sum_{i=1}^r c_i \quad (2.12)$$

An automaton is considered better than the pure-chance automaton if its average penalty $M(t)$ is smaller than M_0 at least asymptotically, as $t \rightarrow \infty$. As $M(t)$ and $\lim_{t \rightarrow \infty} M(t)$ are random variables, one can compare $E[M(t)]$ with M_0 , where

$$E[M(t)] = E[E[\beta(t) | P(t)]] = E[\beta(t)] \quad (2.13)$$

Based on the comparison with the pure-chance automaton, any automaton that performs better than the pure-chance automaton is considered *expedient*. Mathematically, this definition can be expressed as follows:

Definition 2.2: A learning automaton is considered *expedient* if

$$\lim_{t \rightarrow \infty} E[M(t)] < M_0. \quad (2.14)$$

More strictly, the behavior of an automaton can be characterized by the following definition:

Definition 2.3: A learning automaton is said to be *absolutely expedient* if

$$E[M(t+1) | P(t)] < M(t), \quad (2.15)$$

where $M(t)$ is the expected penalty probability at instant 't', and $P(t)$ is the probability vector.

The condition of absolute expediency imposes an inequality on the expected penalty probability $M(t)$ at each instant. Taking expectations again in equation (2.15) we obtain [11]:

$$E[M(t+1)] < E[M(t)]. \quad (2.16)$$

which shows that $E[M(t)]$ is strictly decreasing with 't' in all stationary random environments.

As stated earlier, the goal of any automaton is to learn to asymptotically choose the best action. An automaton that achieves this goal is considered *optimal*. Mathematically, the definition of optimality in the context of learning automata is given as follows:

Definition 2.4: A learning automata is considered *optimal* if

$$\lim_{t \rightarrow \infty} p_b(t) \rightarrow 1 \text{ with probability 1,} \quad (2.17)$$

where $p_b(t)$ is the action probability associated with the minimum penalty probability $c_{\min}$.

Unfortunately, at present, there are no optimal learning automata. In this case, one might aim at a sub-optimal performance, termed as ϵ -optimality [33].

Definition 2.5: Let λ be a learning parameter. A learning automaton is said to be ϵ -optimal if for every $\epsilon > 0$ and $\delta > 0$, there exists $t_0 > \infty$ and $\lambda_0 > 0$ such that

$$\Pr[|p_b(t) - 1| < \epsilon] > 1 - \delta, \quad (2.18)$$

for all $t \geq t_0$ and $\lambda < \lambda_0$.

The ϵ -optimal definition implies that given enough time and given an internal parameter λ (usually depending on the number of internal states), the probability of choosing the best action almost all the time, can be made as close to unity as desired.

It has been shown that if an automaton is *absolute expedient* then, it is also *ϵ -optimal* in all stationary random environments [8].

2.3 Fixed Structure and Variable Structure Learning Automata

A variety of learning automata have been proposed, beginning with Tsetlin's pioneering paper in 1961 [31]. Initial learning automata designs had time invariant transition and output functions, being considered "*fixed structure*" learning automata. Tsetlin, Krylov, and Krinsky [31] [32] presented notable examples of these automata type. *Variable Structure Stochastic Automata* (VSSA) were developed later, in which the state transition functions and the output functions were time dependent [11].

2.3.1 Fixed Structure Automata

2.3.1.1 Tsetlin Automaton

This was the first learning automaton presented in the literature [31]. It is a deterministic fixed structure automaton, denoted $L_{2N,2}$, with $2N$ states and 2 actions, *i.e.* N states for each action. Furthermore, its structure can easily be extended to deal with r ($2 < r < \infty$) actions. As any learning automata, its goal is to incorporate knowledge from the past behavior of the system in its decision rule for choosing the next sequence of actions. To achieve this, the automaton calculates the number of successes and failures received for each action, and switches to the alternate action only when it receives a sufficient number of failures, depending on its current state. In order to describe the

behavior of this type of automaton, its output and state transition functions will be described below.

The output function of the automaton is simple: if the automaton is in a state q_i ($1 \leq i \leq N$), it chooses action α_1 and if it is in a state q_i ($N+1 \leq i \leq 2N$) it chooses action α_2 . Since each action has N states associated with it, N is called the memory associated with each action and the automaton is said to have a total memory of $2N$.

The state transitions are illustrated by the two graphs presented in Figure 2.4, one for a favorable response and one for an unfavorable response. If the environment replies with a reward (favorable response), the automaton moves deeper into the memory of the corresponding action. If the environment replies with a penalty (unfavorable response), the automaton moves towards the outside boundary of the memory of the corresponding action. The deepest states in memory are referred to as the most internal states, or the end states.

Figure 2.4: State transition graphs for the Tsetlin automaton $L_{2N,2}$

The Tsetlin automaton can be analyzed using the theory of Markov chains [11]. From this perspective, it is possible to characterize the states of the automaton as *recurrent states*, meaning that the automaton can be in any state an infinite number of times. For example, if the automaton is in state 1, and action 1 is being penalized N consecutive times, and after that the automaton is rewarded N consecutive times, the automaton will move from state 1 to state $N+1$. Similarly, the automaton will move from the state $N+1$ to the state 1. Furthermore, the automaton is *irreducible* because every two states (q_i, q_j) communicate [11]. Another characteristic of this automaton is that it is *aperiodic*, since it can loop in the states 1 and $N+1$ an arbitrary number of times.

Any finite Markov chain that is irreducible and aperiodic is ergodic [6], and this implies that the Tsetlin automaton is *ergodic*. This property of the Tsetlin automaton indicates that the automaton will converge to a state probability distribution independently of the probability distribution at the starting state.

The expected asymptotic penalty probability for the Tsetlin automaton was shown to be

$$M^*(L_{2N,2}) = \frac{\frac{1}{c_1^{N-1}} \cdot \frac{c_1^N - d_1^N}{c_1 - d_1} + \frac{1}{c_2^{N-1}} \cdot \frac{c_2^N - d_2^N}{c_2 - d_2}}{\frac{1}{c_1^N} \cdot \frac{c_1^N - d_1^N}{c_1 - d_1} + \frac{1}{c_2^N} \cdot \frac{c_2^N - d_2^N}{c_2 - d_2}}, \quad (2.19)$$

where c_i is the penalty probability of α_i , $c_i = 1 - d_i$, $i=1,2$ [11]. It has been shown that the Tsetlin automaton is ϵ -optimal in all environments whenever $\min\{c_1, c_2\} \leq 0.5$ [11].

In 1964, Krinsky made an important step in the learning automata theory by presenting another deterministic automaton, which was ϵ -optimal in all environments. The next section presents this automaton in detail.

2.3.1.2 Krinsky Automaton

This automaton, denoted $K_{2N,2}^1$, is a deterministic automaton, and like its predecessor, the Tsetlin automaton, is an automaton with $2N$ states and 2 actions. The output function is identical to the output function of the Tsetlin automaton, *i.e.* if the automaton is in any state q_i ($i=1,2,\dots,N$) it chooses action α_1 , and if it is in any state q_i ($i=N+1,N+2,\dots,2N$), it chooses action α_2 .

The state transition function of Krinsky automaton is similar but not identical to the state transition function of the Tsetlin automaton. When the environment replies with a penalty, the automata exhibit the same behavior, *i.e.* they move towards the outside margins of their current action's domain. The difference lies in the way the automata react when the environment rewards an action. Tsetlin had his automata move exactly one state closer or further from its internal states for each reward or penalty. The “philosophy” behind Krinsky’s automaton is to give a maximum effect for each reward. This implies moving to the deepest state in the memory when the environment rewards an action, and N consecutive penalties are required to change the action of the automaton. In the case of a favorable response, if the automaton is in any state q_i ($i=1,2,\dots,N$), it passes to state q_1 and if it is in any state q_i ($i=N+1,N+2,\dots,2N$), it passes to the state q_{N+1} , as shown in Figure 2.5.

Figure 2.5: State transition graphs for the Krinsky automaton $K'_{2N,2}$

The expected asymptotic penalty probability for the Krinsky automaton was calculated to:

$$M^*(K_1) = \frac{\frac{1}{c_1^{N-1}} + \frac{1}{c_2^{N-1}}}{\frac{1}{c_1^N} + \frac{1}{c_2^N}}, \quad (2.20)$$

where c_i , $i=1,2$ is the penalty probability [11]. It can be easily shown that the Krinsky automaton is ϵ -optimal in all stationary random environments [11].

The Tsetlin and Krinsky automata are considered deterministic automata, since both their output, and state transition functions are deterministic. In order to present the whole class of fixed structure automata, the next section presents a learning automaton that is stochastic, introduced by Krylov in 1964.

2.3.1.3 Krylov Automaton

Krylov automaton ($K^2_{2N,2}$) is also an automaton with $2N$ states and 2 actions, and has the same output transition function as the $L_{2N,2}$ automaton. Furthermore, Krylov's automaton has the same state transition function as $L_{2N,2}$ automaton but only when the response of the environment is favorable. The behavior differs when the automaton is penalized; in this situation, the behavior of the Krylov automaton is stochastic rather than deterministic. If the automaton is penalized, it moves towards or outwards it's internal states with a probability 0.5, as shown in the Figure 2.6.

Figure 2.6: State transition graphs for the Krylov automaton $K^2_{2N,2}$

It is important to note that the modification made by Krylov makes the automaton ϵ -optimal in all environments. The expected asymptotic penalty probability is shown to be:

$$M^*(K_2) = \frac{\frac{1}{\lambda_1^{N-1}} \cdot \frac{\lambda_1^N - 1}{\lambda_1 - 1} + \frac{1}{\lambda_2^{N-1}} \cdot \frac{\lambda_2^N - 1}{\lambda_2 - 1}}{\frac{1}{c_1} \cdot \frac{1}{\lambda_1^{N-1}} \cdot \frac{\lambda_1^N - 1}{\lambda_1 - 1} + \frac{1}{c_2} \cdot \frac{1}{\lambda_2^{N-1}} \cdot \frac{\lambda_2^N - 1}{\lambda_2 - 1}}, \quad (2.21)$$

where $\lambda_i = \frac{c_i}{1 - c_i}$, $i = 1, 2$. As N increases, the limit becomes

$$\lim_{N \rightarrow \infty} M^*(K_2) = \min\{c_1, c_2\} \quad (2.22)$$

which proves that the automaton is ε -optimal [11].

Krylov's automaton is a *fixed structure stochastic automaton* (FSSA). The concepts of the automata presented above can be extended to cases where the automata can perform r ($2 \leq r < \infty$) actions $\{\alpha_1, \alpha_2 \dots \alpha_r\}$. The automata with many actions and the automata with only two actions differ mainly in those states where the automaton switches from one action to the next. A detailed description of this generalization can be found in [32] [11].

2.3.2 Variable Structure Stochastic Automata

In search for a greater flexibility for designing automata, Varshavskii and Vorontsova [34] were the first to propose a class of automata that update transition probabilities; these are called *Variable Structure Stochastic Automata* (VSSA). The principal characteristic of this type of automata is that the state transition probabilities or the action selecting probabilities are updated with time.

For mathematical simplicity, it is assumed that each state corresponds to a distinct action. This implies that the number of states s is equal to the number of actions r ($s = r < \infty$) and so, the action transition mapping G becomes the identity mapping. Varshavskii and Vorontsova have proved that every VSSA is completely defined by a set of action probability updating rules, and so the state transition mapping F becomes

equivalent to the probability updating rule for the definition of a VSSA. The learning automata operates on a probability vector $\mathbf{P}(t)=[p_1(t), \dots, p_r(t)]^T$, where $p_i(t)$ ($i=1, \dots, r$) is the probability that the automaton will select the action α_i at the time t : $p_i(t)=\Pr[\alpha(t)=\alpha_i]$. A mathematical description of a variable structure stochastic automaton is given below:

Definition 2.6: A *variable structure stochastic automaton* (VSSA) is a 4-tuple $\langle A, B, T, \mathbf{P} \rangle$, where A is the set of actions, B is the set of inputs of the automaton (the set of outputs of the environment), and $T:[0,1]^r \times B \rightarrow [0,1]^r$ is an updating scheme such that

$$\mathbf{P}(t+1) = T(\mathbf{P}(t), \beta(t)), \quad (2.23)$$

where $\mathbf{P}$ is the action probability vector, $\mathbf{P}(t)=[p_1(t), p_2(t), \dots, p_r(t)]^T$, with $p_i(t)=\Pr[\alpha(t)=\alpha_i]$, $i=1, \dots, r$, and $\sum_{i=1}^r p_i(t) = 1$ for all 't'.

As in the case of FSSA, the VSSA can be analyzed using the Markov chain theory. If the mapping T is independent of time, the probability $\mathbf{P}(t+1)$ is determined completely by $\mathbf{P}(t)$, which implies that $\{\mathbf{P}(t)\}_{t \geq 0}$ is a discrete-homogenous Markov process. From this perspective, different mappings T can identify different types of learning algorithms. If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as *absorbing algorithm*. Similarly, *non-absorbing algorithms* are Markov processes with no absorbing states. Ergodic VSSA are suitable for non-stationary environments because their behavior is independent of their initial states. Thathachar and Narendra have presented different varieties of absorbing algorithms in [11]. Ergodic VSSA have been proposed in [11],

[12], [7]. The goal of a VSSA is to choose a mapping T such that the learning algorithm satisfies one of the performance criteria.

The VSSA can be classified according to the updating probability functional form. If $P(t+1)$ is a linear function of $P(t)$, the automaton is said to be *linear*, otherwise it is considered *nonlinear*. Occasionally, two or more automata are combined to form a *hybrid* automaton. Independent of the form of the updating scheme, a VSSA follows some basic learning principles. If an action α_i has been rewarded, the automaton increases the probability for this action, decreasing the probability for all other actions. If an action α_i has been penalized, the automaton decreases the probability for this action, increasing the probability for all other actions. Depending on the learning principle of its VSSA, different combinations of updating schemes can be enumerated as:

- **RP (Reward-Penalty)** – the probabilities are updated when the automaton is rewarded and penalized.
- **RI (Reward-Inaction)** – the probabilities are updated when the automaton is rewarded and are left unchanged when the automaton is penalized.
- **IP (Inaction-Penalty)** – the probabilities are updated when the automaton is penalized and are left unchanged when the automaton is rewarded.

If the mapping T is a continuous one, the automaton is considered a *continuous automaton*. The VSSA presented originally were continuous algorithms. In 1979, Thathachar and Oommen introduced discretized versions of learning VSSA [26], which have been later extended to yield varieties of absorbing, ergodic and estimator type of learning automata [14] [17] [19] [18].

A general updating scheme for a continuous VSSA operating in a stationary environment with $\beta = \{0, 1\}$ can be represented as follows:

If action α_i is chosen $\alpha(t) = \alpha_i$, the updated probabilities are:

$$\begin{aligned} p_j(t+1) &= p_j(t) - g_i(P(t)), & \text{when } \beta(t) = 0 \\ p_j(t+1) &= p_j(t) + h_j(P(t)), & \text{when } \beta(t) = 1 \end{aligned}, \text{ for all } j, j \neq i. \quad (2.24)$$

Because $P(t)$ is a probability vector, it has to satisfy $\sum_{j=1}^r p_j(t) = 1$, which implies that

$$\begin{aligned} p_i(t+1) &= p_i(t) + \sum_{\substack{j=1 \\ j \neq i}}^r g_j(P(t)), & \text{when } \beta(t) = 0 \\ p_i(t+1) &= p_i(t) - \sum_{\substack{j=1 \\ j \neq i}}^r h_j(P(t)), & \text{when } \beta(t) = 1 \end{aligned} \quad (2.25)$$

In the above representation, the functions h_j and g_j have the following properties:

- h_j and g_j are continuous functions (assumed for mathematical convenience [11])
- h_j and g_j are nonnegative functions,

$$0 < g_j(P(t)) < p_j(t)$$

$$\bullet \quad 0 < \sum_{\substack{j=1 \\ j \neq i}}^r [p_j(t) + h_j(P(t))] < 1,$$

for all $i=1, 2, \dots, r$ and all $P(t)$ whose elements are in the open interval $(0, 1)$. As mentioned above, if the functions h_j and g_j are linear in $P(t)$, the automata are said to be *linear*.

VSSA are implemented using a Random-Number Generator (RNG). The automaton decides on which action to choose based on the action probability distribution $P(t)$.

From the class of linear VSSA, the following three algorithms are relevant because they express the three main philosophies of learning: the linear reward-penalty scheme (L_{RP}), the linear reward-inaction scheme (L_{RI}) and the linear inaction-penalty scheme (L_{IP}) scheme. All these schemes are explained below for learning automata with two actions. Their extension to multiple actions can be found in [11].

2.3.2.1 Linear Reward-Penalty Scheme (L_{RP})

In a linear reward-penalty scheme, the automaton increases the probability for the action that has been rewarded, and decreases the probability for the action that has been penalized. This method of learning gives the following updating equations:

$$\begin{aligned}
 p_1(t+1) &= p_1(t) + \lambda_1(1 - p_1(t)) & \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 0 \\
 p_1(t+1) &= (1 - \lambda_2)p_1(t) & \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 1 \\
 p_1(t+1) &= (1 - \lambda_1)p_1(t) & \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 0 \\
 p_1(t+1) &= p_1(t) + \lambda_2(1 - p_1(t)) & \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 1
 \end{aligned} \tag{2. 26}$$

where $0 < \lambda_1 < 1$ and $0 < \lambda_2 < 1$ are the reward and penalty parameters, respectively. These equations show that whenever a probability $p_k(t)$ is increased, it is increased with a value proportional with the distance to 1, namely $[1 - p_k(t)]$. When a probability $p_k(t)$ is decreased, it is decreased with a value proportional with its distance to 0, *i.e.*, $p_k(t)$. The specific case when $\lambda_1 = \lambda_2$ it is known as the *symmetric linear reward-penalty scheme* (L_{RP}).

The L_{RP} scheme is ergodic. Also, it was shown that the asymptotic value of the average penalty for the symmetric L_{RP} scheme is given by:

$$\lim_{t \rightarrow \infty} E[M(t)] = \frac{2c_1c_2}{c_1 + c_2} < \frac{c_1 + c_2}{2} = M_0, \quad (2.27)$$

where c_1 and c_2 are the penalty probabilities. This proves that the L_{RP} scheme is expedient for all initial conditions, in all stationary environments. Since the scheme is ergodic, it is suitable for non-stationary environments [11].

2.3.2.2 Linear Reward-Inaction Scheme (L_{RI})

The basic idea of the reward-inaction scheme (L_{RI}) is to keep the probabilities unchanged whenever the environment replies with an unfavorable response. When a favorable response is given, the probability of the action is increased as in the L_{RP} scheme. The updating equations for this scheme can be derived from the L_{RP} scheme by choosing the penalty parameter λ_2 to be 0, and are presented as follows:

$$\begin{aligned} p_1(t+1) &= p_1(t) + \lambda_1(1 - p_1(t)) && \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 0 \\ p_1(t+1) &= p_1(t) && \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 1 \\ p_1(t+1) &= (1 - \lambda_1)p_1(t) && \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 0 \\ p_1(t+1) &= p_1(t) && \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 1 \end{aligned} \quad (2.28)$$

These equations indicate that this scheme has two absorbing states: $[0,1]^T$ and $[1,0]^T$. For example, if $p_1(t)$ becomes unity and action α_1 is rewarded, the probability becomes

$$p_1(t+1) = 1 + \lambda_1(1 - 1) = 1. \quad (2.29)$$

If the automaton is penalized at this time, then the probability remains unchanged since the automaton does not react to penalties, which implies that the state $[1,0]^T$ is an

absorbing state. In an analogous manner, it can be shown that $[0,1]^T$ is an absorbing state. This makes the L_{RI} scheme inappropriate for non-stationary environments. In any stationary environment, the L_{RI} scheme has proved to be ϵ -optimal [7].

2.3.2.3 Linear Inaction-Penalty Scheme (L_{IP})

This scheme is based on the principle that the probabilities are updated only when an action is being penalized and they remain unchanged when an action is rewarded.

This method of learning can be expressed mathematically as follows:

$$\begin{aligned}
 p_1(t+1) &= p_1(t) && \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 0 \\
 p_1(t+1) &= (1 - \lambda_2)p_1(t) && \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 1 \\
 p_1(t+1) &= p_1(t) && \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 0 \\
 p_1(t+1) &= p_1(t) + \lambda_2(1 - p_1(t)) && \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 1
 \end{aligned} \tag{2.30}$$

It has been proved that this automaton is ergodic and expedient [7].

All these schemes can be obtained from the equations (2.26) by giving different values to the learning parameters λ_1 and λ_2 ; for example, the L_{IP} scheme can be obtained from these equations for $\lambda_1=0$ and the L_{RI} scheme can be obtained for $\lambda_2=0$. Lakshmivarahan and Thathachar [11] [7] studied the general behavior of a linear reward-penalty scheme for different parameters λ_1 and λ_2 . They have shown that a scheme based on the equations (2.26) with $\lambda_2 \in (0,1]$ does not have any absorbing states and the nature of convergence of $\{P(t)\}_{t \geq 0}$ is similar to that of the L_{RP} scheme. Furthermore, they have shown that for small values of the parameter λ_2 relative to λ_1 , the equation Eq. (2.26) can generate an ϵ -optimal scheme [11]. This led to a L_{R-EP} scheme which is ergodic and ϵ -

optimal [11]. This scheme was obtained by adding a small penalty term to the L_{RI} scheme; *i.e.* it can be viewed as a L_{RP} scheme where the penalty terms are made small in comparison with the reward terms. The importance of this scheme is that it has a marked ability to be used in non-stationary environments and yet has good convergence properties.

Another method used to improve the convergence of VSSA is to discretize the probability space. The next section describes this method and presents few examples of discretized learning automata.

2.3.3 Discretized Learning Automata

Prior to introducing the concept of discretization, all the existing continuous VSSA permitted action probabilities to take any value in the interval $[0,1]$. In their implementation, the learning automata use Random-Number Generator (RNG) in determining which action to choose. In theory, an action probability can take any value between 0 and 1, so the RNG is required to be very accurate; however, in practice, the probabilities are rounded-off to an accuracy depending on the architecture of the machine that is used to implement the automaton.

In order to increase the speed of convergence of these algorithms and to minimize the requirements of the RNG, the concept of discretizing the probability space was introduced [26] [19]. Analogous to the continuous algorithms, the discretized VSSA can be defined using probability updating functions, but these functions can take values in a discrete finite space. These values divide the continuous $[0,1]$ interval into a finite

number of continuous subintervals. The Discrete Algorithms are said to be *linear* if these subintervals have equal length, otherwise they are called *nonlinear* [26].

Like the continuous learning automata, the discretized learning automata can be analyzed using the theory of Markov chains, and can be divided into two categories: ergodic or absorbing.

Following the discretization concept, many of the continuous variable structure stochastic automata have been discretized. Various discrete automata have been presented in literature [19] [17] [14] [26]. All of the linear VSSA presented in the previous section have corresponding discretized versions: *i.e.* the discretized linear reward-penalty automaton (DL_{RP}), the discretized linear reward-inaction automaton (DL_{RI}), and the discretized linear inaction-penalty automaton (DL_{IP}) [26] [14] [17]. The concepts of discretized automata will be presented in the following sections by demonstrating the similarities and dissimilarities between some continuous automata and their discrete counterparts.

2.3.3.1 Discretized Linear Reward-Inaction Automaton

The discretized linear reward–inaction automaton (DL_{RI}) was the first discretized automaton presented in the literature [26]. In the following description of this automaton, only two actions are considered, but the same concept applies to a r -action ($2 < r < \infty$) automaton. The basic idea of the learning algorithm is to make discrete changes in the action probabilities. The probability space $[0,1]$ is divided into N intervals, where N is a resolution parameter and is recommended to be an even integer. Since it is a reward-

inaction automaton, the updating equations do not modify the action probability vector when the environment penalizes the automaton. When the response from the environment is a reward, the automaton increases the probability of the action that has been chosen and decreases the probability for all the remaining actions.

The discretized automaton has a state associated with every possible probability value, which determines the following set of states: $Q=\{q_1, q_2, \dots, q_N\}$. In every state q_i , the probability that the automaton chooses action α_1 is i/N and the probability to choose action α_2 is $(1 - i/N)$. The state transition map is defined by the following equations:

$$\begin{aligned} q(t+1) &= q_{i+1}, & \text{if } \alpha(t) = \alpha_1 \text{ and } \beta(t) = 0 \\ q(t+1) &= q_{i-1}, & \text{if } \alpha(t) = \alpha_2 \text{ and } \beta(t) = 0 \\ q(t+1) &= q_i, & \text{if } \alpha(t) = \alpha_1 \text{ or } \alpha_2 \text{ and } \beta(t) = 1. \end{aligned} \quad (2.31)$$

where $q(t)=q_i \neq q_0$ or q_N . It can be seen that both q_0 and q_N are absorbing states. Based on the probabilities associated with each state, the automaton can be described entirely by the following action probability updating equations:

$$\begin{aligned} p_1(t+1) &= \min\left\{1, p_1(t) + \frac{1}{N}\right\}, & \text{if } \alpha(t) = \alpha_1, \beta(t) = 0 \\ p_1(t+1) &= \max\left\{0, p_1(t) - \frac{1}{N}\right\}, & \text{if } \alpha(t) = \alpha_2, \beta(t) = 0, \\ p_1(t+1) &= p_1(t), & \text{if } \beta(t) = 1 \\ p_2(t+1) &= 1 - p_1(t+1). \end{aligned} \quad (2.32)$$

The algorithm starts with the initial action probability vector $P(0) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}^T$ and with a resolution parameter N .

These equations indicate that $\{P(t)\}$ behaves like a homogenous Markov chain with two absorbing states: $[1,0]^T$ and $[0,1]^T$. The algorithm has been proven to be ϵ -optimal in all environments [19]. The difference between this algorithm and its continuous version is in the rate of convergence. Oommen and Hansen have performed simulations of the L_{RI} and DL_{RI} automata, and in all scenarios, the DL_{RI} automaton is superior to the L_{RI} automaton [19]. Their studies indicate that when the two automata were made to learn the best action in an environment with $c_1=0.2$ and $c_2=0.6$, in 240 iterations the L_{RI} automaton gave only an expected value of 0.99982. The DL_{RI} scheme gave an expected value of 0.99999 and subsequently the value stayed at unity. If a stopping criterion was used, it was seen that the DL_{RI} automaton reached 0.99 accuracy in 125 iterations and the L_{RI} automaton reached the same accuracy in 135 iterations.

In [14], Oommen compared DL_{RI} with some deterministic automata. The Table 2.1 shows a comparison between the performance of various learning automata.

Table 2.1: Experimental comparative performance of DL_{RI} with other FSSA [14].

c_1	c_2	N	$E[p_1(\infty)]$	DL_{RI} Mean Time For Convergence (No. iterations)	Tsetlin $p_1(\infty)$	Krinsky $p_1(\infty)$
0.4	0.8	4	0.84	6	0.80	0.80
		10	0.95	16	0.95	0.97
		20	0.99	31	0.99	0.99
0.6	0.8	4	0.74	9	0.64	0.64
		10	0.86	26	0.72	0.81
		20	0.94	53	0.75	0.95
0.7	0.8	4	0.65	11	0.57	0.57
		10	0.73	34	0.60	0.66
		20	0.81	76	0.60	0.79

From these results, Oommen concluded that for environments with $c_1 > 0.5$, the DL_{RI} is more accurate than the Tsetlin automaton. Furthermore, for a fixed N, as the difference between the penalty probabilities is decreased, DL_{RI} becomes more accurate than the Tsetlin and Krinsky automata. In all these environments, Oommen observed that the DL_{RI} automaton is faster than the Tsetlin and Krinsky automata. It was later shown that the DL_{RI} scheme is ϵ -optimal in all random environments [14].

2.3.3.2 Discretized Linear Inaction-Penalty Automaton

Following the same method of discretization used for the reward-inaction algorithm, a discretized version of the linear inaction-penalty algorithm, denoted DL_{IP} , was developed [14]. This algorithm has been proved *ergodic* and *expedient* in all random

environments [14]. A later artificially created absorbing version of this algorithm, the absorbing discretized linear inaction-penalty automaton, denoted ADL_{IP} , was the first inaction-penalty algorithm proved to be ϵ -optimal [14]. Although the ADL_{IP} automaton is ϵ -optimal, simulation results have shown that this scheme is very accurate but slow in convergence [14]. When the penalty probabilities are high, the automaton utilizes many more responses of the environment than a reward-inaction automaton. Table 2.2 presents some comparative results between the ADL_{IP} , the DL_{RI} and the ADL_{RP} automata.

The updating rules for the DL_{IP} algorithm are defined in the following equations:

$$\begin{aligned}
 p_1(t+1) &= \min\left\{1, p_1(t) + \frac{1}{N}\right\}, & \text{if } \alpha(t) = \alpha_2, \beta(t) = 1 \\
 p_1(t+1) &= \max\left\{0, p_1(t) - \frac{1}{N}\right\}, & \text{if } \alpha(t) = \alpha_1, \beta(t) = 1 \\
 p_1(t+1) &= p_1(t), & \text{if } \alpha(t) = \alpha_1 \text{ or } \alpha_2, \beta(t) = 0 \\
 p_2(t+1) &= 1 - p_1(t).
 \end{aligned} \tag{2.33}$$

2.3.3.3 Discretized Linear Reward-Penalty Automaton

The discretized linear reward-penalty automaton (DL_{RP}), as its continuous version, reacts to both reward and penalty responses of the environment. Similarly to the DL_{RI} and the DL_{IP} automata, DL_{RP} updates its action probabilities in steps of size $1/N$, where N is a resolution parameter. The updating rules for this automaton are given by the following equations:

$$\begin{aligned}
p_1(t+1) &= \min\left\{1, p_1(t) + \frac{1}{N}\right\}, \text{ if } \alpha(t) = \alpha_1, \beta(t) = 0 \text{ or } \alpha(t) = \alpha_2, \beta(t) = 1 \\
p_1(t+1) &= \max\left\{0, p_1(t) - \frac{1}{N}\right\}, \text{ if } \alpha(t) = \alpha_1, \beta(t) = 1 \text{ or } \alpha(t) = \alpha_2, \beta(t) = 0 \quad (2.34) \\
p_2(t+1) &= 1 - p_1(t+1).
\end{aligned}$$

Oommen and Christensen [17] proved that the DL_{RP} automaton is ergodic and ϵ -optimal in all random environments whenever $c_{min} < 0.5$. They also showed that by making a stochastic modification to the transition function, the automata can be made ergodic and ϵ -optimal in all random environments. This modified version of the DL_{RP} automaton is known as the *modified discrete linear reward-penalty automaton*, MDL_{RP} , and is the only known ergodic linear reward-penalty scheme, which is ϵ -optimal in all random environments. Oommen and Christensen created an absorbing version of the DL_{RP} automaton denoted ADL_{RP} . They showed that a discretized two-action linear reward-penalty automaton with artificially created absorbing barriers is ϵ -optimal in all random environments. It is the only symmetric ϵ -optimal learning automata known. Simulation results indicated that the ADL_{RP} scheme is extremely accurate and fast in convergence [17].

Oommen and Christensen have also made a comparative study of the performance and accuracy of some of the discrete linear automata [17]. The results of this study, presented in Table 2.2, show that the ADL_{RP} scheme is superior based on counts of both speed and accuracy.

Table 2.2: Comparative performance of DL_{RI} , ADL_{IP} , ADL_{RP} ($c_2=0.8$)

N	c_1	DL_{RI}		ADL_{IP}		ADL_{RP}	
		$E[p_1(\infty)]$	M.T.C.	$E[p_1(\infty)]$	M.T.C.	$E[p_1(\infty)]$	M.T.C.
4	0.2	0.896	4.61	0.960	9.87	0.95	2.965
	0.4	0.843	6.04	0.825	10.65	0.86	3.730
	0.6	0.741	8.52	0.655	10.57	0.72	4.830
10	0.2	0.980	11.46	1.00	70.28	1.00	8.220
	0.4	0.951	16.15	1.00	196.78	0.98	14.88
	0.6	0.855	25.58	0.93	499.11	0.93	32.45

These results indicate, for example, that if $N=10$, $c_1=0.6$ and $c_2=0.8$, the DL_{RI} scheme converges with an expected accuracy of 0.855, and the mean time to converge (M.T.C.) was 25.58 iterations. With the same parameters, the ADL_{IP} scheme converged with a greater accuracy (0.93) but the mean time to converge was much bigger, 499.11 iterations. The results for the ADL_{RP} have shown that it converged with an accuracy of 0.93 and the mean time to converge was 32.45 iterations. The following table summarizes the convergence characteristics of all the VSSA, in their continuous and discrete forms.

Table 2.3: Comparison between continuous and discrete linear VSSA

Algorithm	Type	Markov Chain Characterization	Convergence Behavior
L_{RI}	continuous	absorbing	ϵ -optimal in all environments
DL_{RI}	discrete	absorbing	ϵ -optimal in all environments
L_{IP}	continuous	ergodic	expedient
DL_{IP}	discrete	ergodic	expedient
ADL_{IP}	discrete	absorbing	ϵ -optimal in all environments
L_{RP}	continuous	ergodic	expedient in all stationary env.
DL_{RP}	discrete	ergodic	ϵ -optimal if $c_{min} < 0.5$
ADL_{RP}	discrete	absorbing	ϵ -optimal in all environments
MDL_{RP}	discrete	ergodic	ϵ -optimal in all environments

Although in this section we presented only linear schemes, it is important to note that discrete nonlinear schemes have also been developed [14]. A description of these is omitted here in the interest of brevity. The next section presents a new category of learning algorithms, the estimator algorithms.

2.4 Estimator Algorithms

2.4.1 Overview

In the quest to design faster converging learning algorithms, Thathachar and Sastry have opened another path by introducing a new class of algorithms, called estimator algorithms [28]. The main feature of these algorithms is that they maintain running estimates for the penalty probability of each possible action and use them in the probability updating equations. The purpose of these estimates is to crystallize the confidence in the reward capabilities of each action. In their characteristics, these algorithms model the behavior of a person that is trying to choose an action in a random environment. In this task, the most common and simple approach is to try each action a number of times and to estimate the probability of reward for each action. The person will most likely choose the action that has the highest reward estimate; however, unlike straightforward estimation, the superior actions are also chosen in a more likely manner in the estimation process.

From this perspective, all the algorithms presented in the previous sections are non-estimator algorithms. The main difference between the estimator algorithms and the non-estimator algorithms lies in the way the action probability vector is updated. The non-estimator algorithms update the probability vector based directly on the response of the environment. If the chosen action is rewarded, then the automaton increases the

probability of choosing this action at the next time instant. Otherwise, the action probability of the selected action is decreased.

The estimator algorithms are characterized by the use of the estimates for each action. The change of the probability of choosing an action is based on its current estimated mean reward, and possibly on the feedback of the environment. The environment determines the probability vector indirectly, through the calculation of the reward estimates for each action. Even when the chosen action is rewarded, there is a possibility that the probability of choosing another action is increased.

For the definition of an estimator learning automaton, a vector of reward estimates $\hat{\mathbf{d}}(t)$ must be introduced. Hence, the state vector $\mathbf{Q}(t)$ is defined as $\mathbf{Q}(t) = \langle \mathbf{P}(t), \hat{\mathbf{d}}(t) \rangle$, where $\hat{\mathbf{d}}(t) = [\hat{d}_1(t) \dots \hat{d}_r(t)]^T$ [29]

Thathachar and Sastry have shown that the estimator algorithms exhibit a superior speed of convergence when compared with the non-estimator algorithms [29]. In 1989, Oommen and Lanctôt introduced discretized versions of the estimator algorithms, and have shown that the discretized estimator algorithms are even faster than their continuous counterparts [10].

2.4.2 Continuous Estimator Algorithms

This section describes the class of continuous estimator algorithms.

2.4.2.1 Pursuit Algorithm

Thathachar and Sastry [27] introduced the concept of estimator algorithms by presenting a Pursuit Algorithm that implemented a reward-penalty learning philosophy, denoted CP_{RP} . As its name reveals, this algorithm is characterized by the fact that it pursues the action that is currently estimated to be the optimal action. The algorithm achieves this by increasing the probability of the current optimal action if the chosen action was either rewarded or penalized by the environment.

The CP_{RP} algorithm involves three steps [27]. The first step consists of choosing an action $\alpha(t)$ based on the probability distribution $\mathbf{P}(t)$. Whether the automaton is rewarded or penalized, the second step is to increase the component of $\mathbf{P}(t)$ whose reward estimate is maximal (the current optimal action), and to decrease the probability of all the other actions. The probability of the current optimal action is increased with a value direct proportional with the distance to the unity, namely $1-p_m(t)$. All the other probabilities are decreased proportionally with the distance to zero, *i.e.* $p_i(t)$. The last step is to update the running estimates for the probability of being rewarded. For calculating the estimates, two more vectors are introduced: $\mathbf{W}(t)$ and $\mathbf{Z}(t)$, where $Z_i(t)$ is the number of times the i^{th} action has been chosen and $W_i(t)$ is the number of times the

action α_i has been rewarded. Then, the estimate vector $\hat{\mathbf{d}}(t)$ can be calculated using the following formula:

$$\hat{d}_i(t) = \frac{W_i(t)}{Z_i(t)} \text{ for } i=1,2,\dots,r \quad (2.35)$$

Since the rest of the thesis will deal with Pursuit and estimator algorithms, we formally present the algorithm below.

ALGORITHM CP_{RP}

Parameters

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

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

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

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

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking each action a small number of times.

Repeat

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

Step 2: If α_m is the current optimal action, update $\mathbf{P}(t)$ according to the following equations:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \lambda \cdot (1 - p_m(t)) \\ p_j(t+1) &= p_j(t) - \lambda \cdot p_j(t) \end{aligned} \quad (2.36)$$

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

$$\begin{aligned} W_i(t+1) &= W_i(t) + (1 - \beta(t)) \\ Z_i(t+1) &= Z_i(t) + 1 \\ \hat{d}_i(t+1) &= \frac{W_i(t+1)}{Z_i(t+1)} \end{aligned} \quad (2.37)$$

$$\left. \begin{aligned} W_j(t+1) &= W_j(t) \\ Z_j(t+1) &= Z_j(t) \\ \hat{d}_j(t+1) &= \hat{d}_j(t) \end{aligned} \right\}, \text{ for all } j \neq i \quad (2.38)$$

End Repeat

END ALGORITHM CP_{RP}

The CP_{RP} algorithm is similar in design to the L_{RP} algorithm, in the sense that both algorithms modify the action probability vector $\mathbf{P}(t)$ if the response from the environment is a reward or a penalty. The difference occurs in the way they approach the solution. The L_{RP} algorithm moves $\mathbf{P}(t)$ in the direction of the most recently rewarded action or in the direction of all the actions not penalized, whereas the CP_{RP} algorithm moves $\mathbf{P}(t)$ in the direction of the action which has the highest reward estimate. Thathachar and Sastry proved that this algorithm is ϵ -optimal in every stationary environment. Also, comparing the performance of the CP_{RP} and L_{RI} automata, the authors have shown that the CP_{RP} algorithm converges up to seven times faster than the L_{RI} automaton [27].

2.4.2.2 TSE Algorithm

Thathachar and Sastry in [28] introduced a more sophisticated estimator algorithm, which we refer to as the TSE Algorithm. Being an estimator algorithm, it considers the reward estimates in calculating the action probability vector. The algorithm increases the probabilities for all the actions that have a higher estimate than the estimate of the chosen action, and decreases the probabilities of all the actions with a smaller

estimate. The probabilities are updated based on both the reward estimates $\hat{d}(t)$ and the action probability vector $P(t)$, as shown below, in the detailed of this algorithm.

ALGORITHM TSE

Parameters

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

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

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

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

$S_{ij}(t)$ an indicator function $S_{ij}(t) = \begin{cases} 1 & \text{if } \hat{d}_i(t) > \hat{d}_j(t) \\ 0 & \text{if } \hat{d}_i(t) \leq \hat{d}_j(t) \end{cases}$

f $f: [-1, 1] \rightarrow [-1, 1]$ a monotonic, increasing function satisfying $f(0)=0$

Method

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

Initialize $\hat{d}(t)$ by picking each action a small number of times.

Repeat

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

Step 2: Update $P(t)$ according to the following equations:

$$\begin{aligned} p_j(t+1) &= p_j(t) - \lambda \cdot [f(\hat{d}_i(t) - \hat{d}_j(t))] \cdot \left[S_{ij}(t)p_j(t) + S_{ji}(t) \frac{p_i(t)}{r-1} (1 - p_j(t)) \right], \quad j \neq i \\ p_i(t+1) &= p_i(t) + \lambda \cdot \sum_{j \neq i} [f(\hat{d}_i(t) - \hat{d}_j(t))] \cdot \left[S_{ij}(t)p_j(t) + S_{ji}(t) \frac{p_i(t)}{r-1} (1 - p_j(t)) \right] \end{aligned} \quad (2.39)$$

Step 3: Update $\hat{d}(t)$ according to the following:

$$\begin{aligned} W_i(t+1) &= W_i(t) + (1 - \beta(t)) \\ Z_i(t+1) &= Z_i(t) + 1 \\ \hat{d}_i(t+1) &= \frac{W_i(t+1)}{Z_i(t+1)} \end{aligned} \quad (2.40)$$

$$\left. \begin{aligned} W_j(t+1) &= W_j(t) \\ Z_j(t+1) &= Z_j(t) \\ \hat{d}_j(t+1) &= \hat{d}_j(t) \end{aligned} \right\}, \text{ for all } j \neq i \quad (2.41)$$

End Repeat
END ALGORITHM TSE

It is important to notice that $P(t)$ depends indirectly on the response of the environment. The feedback from the environment changes the values of the reward estimate vector, which affects the values of the functions f and S_{ij} .

The detailed description of the algorithm indicates that if the i^{th} action is rewarded, all the probabilities $p_j(t)$ that correspond to actions with reward estimates higher than the reward estimate $\hat{d}_i(t)$ are updated using the following equation:

$$p_j(t) = p_j(t) - \lambda \cdot f(\hat{d}_i(t) - \hat{d}_j(t)) \cdot \frac{p_i(t)}{r-1} \cdot (1 - p_j(t)). \quad (2.42)$$

Since $\hat{d}_i(t) < \hat{d}_j(t)$, the sign of the function $f(\hat{d}_i(t) - \hat{d}_j(t))$ is negative, and thus, the probability $p_j(t)$ increases proportional to $(1 - p_j(t))$.

For all the actions with reward estimates smaller than $\hat{d}_i(t)$, the probabilities are updated based on the following equation:

$$p_j(t+1) = p_j(t) - \lambda \cdot f(\hat{d}_i(t) - \hat{d}_j(t)) \cdot p_j(t). \quad (2.43)$$

The sign of the function $f(\hat{d}_i(t) - \hat{d}_j(t))$ is positive, which means that the action probability $p_j(t)$ is decreased proportionally to $p_j(t)$.

The probability action $p_i(t)$ is increased or decreased to ensure that the sum of all the action probabilities is 1. When all the reward estimates are higher than the reward estimate of action α_i , the automaton increases all the probabilities $p_j(t+1)$. To ensure that the amount that this probabilities increase does not surpass the value of $p_i(t)$, Thathachar and Sastry introduced the term $\frac{p_i(t)}{r-1}$ in the updating equations.

There are two main differences between the Pursuit Algorithm and the TSE Algorithm. First, they differ in the method of deciding which probability actions are increased and which are decreased. The Pursuit algorithm increases only the probability of the action corresponding to the highest estimate, whereas the TSE Algorithm increases the probabilities for all the actions with a higher reward estimate than the estimate of the chosen action. Second, their updating equations differ. In increasing or decreasing a probability, the TSE Algorithm considers also the distance between estimates, incorporated in the term $f(\hat{d}_i(t) - \hat{d}_j(t))$, whereas the Pursuit Algorithm takes into account only the distance between the probability at time t and the probability that it aims for each action, 0 or 1.

This algorithm has been shown to be ϵ -optimal [29]. Also, Thathachar and Sastry presented simulation results in comparison with the L_{RI} scheme. They have shown that for the same level of accuracy, the TSE Algorithm often converges at least seven times faster than the L_{RI} scheme.

2.4.3 Discrete Estimator Algorithms

The Discrete Estimator Algorithms (DEA) were introduced as an approach to create even faster converging learning algorithms [9] [10]. They emerged from applying the discretization “philosophy” to the existing estimator algorithms. In this way, the action probabilities are allowed to take a finite set of values and they use the reward estimates in their updating rules.

Lanctôt and Oommen have defined a set of properties that every discrete estimator algorithm must possess [9] [10]. These properties are known as the *Property of Moderation* and the *Monotone Property*.

Property 1: A DEA with r action and a resolution parameter N is said to possess the *property of moderation* if the maximum magnitude by which an action probability can decrease per iteration is bounded by $1/rN$.

The monotone property can be stated as below:

Property 2: Suppose there exists an index m and a time instant $t_0 < \infty$, such that $\hat{d}_m(t) > \hat{d}_j(t)$ for all $j \neq m$ and all $t \geq t_0$. A DEA is said to possess the *Monotone Property* if there exists an integer N_0 such that for all resolution parameters $N > N_0$, $p_m(t) \rightarrow 1$ with probability one as $t \rightarrow \infty$.

This means that if the estimate of reward of an action α_m remains the maximum estimate after a certain point in time, then a DEA has the monotone property if it steadily increases the probability of choosing α_m to unity.

These properties are necessary in proving that a discretized estimator algorithm is *ϵ -optimal*. Lanctôt and Oommen have proved that any discretized estimator algorithm possessing both of these properties is *ϵ -optimal* [9] [10].

The discrete versions of the Pursuit Algorithm and the TSE Algorithm are presented in the next sections of this chapter.

2.4.3.1 Discrete Pursuit Algorithm

In 1989, Lanctôt and Oommen introduced a discretized version of the Pursuit Algorithm [9] based on the reward-inaction learning “philosophy”, denoted DP_{RI} . The differences between the discrete and continuous version of the Pursuit algorithms occur only in the updating rules for the action probabilities. The discrete Pursuit algorithm makes changes to the probability vector $\mathbf{P}(t)$ in discrete steps, whereas the continuous version uses a continuous function to update $\mathbf{P}(t)$. Being a reward-inaction algorithm, the action probability vector $\mathbf{P}(t)$ is updated only when the current chosen action was rewarded. If the current action is penalized, the action probability vector $\mathbf{P}(t)$ remains unchanged, which implies that the algorithm uses the estimates in updating the action probability vector $\mathbf{P}(t)$ only if the environment rewards the chosen action. When the chosen action is rewarded, the algorithm decreases the probability for all the actions that do not correspond to the highest estimate, by a step Δ , where $\Delta=1/rN$. In order to keep the sum of the components of the vector $\mathbf{P}(t)$ equal to unity, the DP_{RI} increases the probability of the action with the highest estimate by an integral multiple of the smallest step size Δ . A description of the algorithm is given below:

ALGORITHM DP_{RI}**Parameters**

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

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

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

N resolution parameter

Δ $\Delta = 1/rN$ is the smallest step size

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking each action a small number of times.

Repeat

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

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

If $\beta(t) = 0$ and $p_m(t) \neq 1$

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

$$p_m(t+1) = 1 - \sum_{j \neq m} p_j(t+1) \quad (2.44)$$

Else

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

Step 3: Update the reward estimate vector $\hat{\mathbf{d}}(t)$ (same as in the CP_{RP})

End Repeat**END ALGORITHM DP_{RI}**

Oommen and Lanctôt proved that this algorithm satisfies both the properties of moderation and monotonically [18]. Also, they have shown that the algorithm is ϵ -optimal in every stationary random environment [18].

Oommen and Lanctôt performed simulation of the DP_{RI} in some benchmark environments and the results have been compared against the results of the CP_{RP} algorithm. The results have shown that in some difficult environments, the DP_{RI} requires

only 50% of the number of iterations required for its continuous version. In a ten-action environment, the DP_{RI} algorithm required 69% of the iterations required by the CP_{RI} [18].

2.4.3.2 Discrete TSE Algorithm

Oommen and Lanctôt also presented the discretized version of the TSE Algorithm, denoted DTSE [10]. As the authors have said, “the design of this algorithm 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” [10].

Oommen and Lanctôt have justified the transformation of the TSE Algorithm into a discretized one, by analyzing each factor that is part of the updating rules of the TSE Algorithm. The parameter λ , representing the maximum that the continuous probability component can change, has been replaced by the integer θ . The term $f(\hat{d}_i(t) - \hat{d}_j(t))$ was preserved in the DTSE Algorithm, representing a factor of the difference between the reward estimates. The term $\left(S_{ij}(t)p_i(t) + S_{ij}(t)\frac{p_i(t)}{r-1}(1-p_j(t)) \right)$ has been transformed in $\left(S_{ij}(t) + S_{ji}(t)\frac{1}{r-1} \right)$ by eliminating the continuous dependency on the probability vector, as part of the discretization process.

The resultant factor $\theta \cdot f(\hat{d}_i(t) - \hat{d}_j(t)) \left(S_{ij}(t) + S_{ji}(t)\frac{1}{r-1} \right)$ determines the maximum value that a probability will be increased or decreased. In order to make these

changes discrete, the above factor has to be represented in terms of the number of Δ -steps that preserve also the probabilities in the interval $[0,1]$. To do this, two new functions have been introduced: $\text{Rnd}(x)$ and $\text{Check}(p_i(t), p_j(t), x)$. The $\text{Rnd}()$ function rounds up its parameter so that its value is always an integer. The $\text{Check}()$ function calculates the largest integer multiple of Δ , between 1 and x that can be added to $p_i(t)$ and subtracted from $p_j(t)$, and which simultaneously preserves these probabilities in the interval $[0,1]$.

The algorithm modifies first the value of the action probability that has the highest reward estimate to guarantee that this value will always increase. A description of the algorithm is given bellow.

ALGORITHM Discrete TSE

Parameters

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

$\Delta = 1/rN\theta$, with θ (an integer) being the maximum any component can change by.

$\text{Rnd}(x)$ 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 $w \leq x$ such that

$$0 \leq p_i(t) + w\Delta, \quad p_j(t) - w\Delta \leq 1.$$

Method

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

Initialize $\hat{d}(t)$ by picking each action a small number of times.

Repeat

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

Step 2: Update $P(t)$ according to the following:

For each action j , starting with m Do

$$\text{change} = \text{Rnd} \left(\theta \cdot f(\hat{d}_i(t) - \hat{d}_j(t)) \cdot \left(S_{ij}(t) + S_{ji}(t) \frac{1}{r-1} \right) \right)$$

$$\begin{aligned}
 p_j(t+1) &= p_j(t) - \Delta \cdot \text{Check}(p_i(t), p_j(t), \text{change}) \\
 p_i(t+1) &= p_i(t) + \Delta \cdot \text{Check}(p_i(t), p_j(t), \text{change})
 \end{aligned}
 \tag{2.45}$$

EndFor

Step 3: Same as in the TSE Algorithm.

End Repeat

END ALGORITHM Discrete TSE

Although the algorithm is a complicated one, Oommen and Lanctôt proved that it possesses both the moderation and the monotony properties which makes it ε -optimal [18].

The authors have also performed simulations in order to study the performance of the discrete TSE in comparison to its continuous version. Some of the results are presented in the following table:

Table 2.4: The number of iterations until convergence in two-action environments for the TSE Algorithm [10]

Probability of Reward		Mean Iterations	
Action 1	Action 2	Continuous	Discrete
0.800	0.200	28.8	24.0
0.800	0.400	37.0	29.0
0.800	0.600	115.0	76.0
0.800	0.700	400.0	380.0
0.800	0.750	2200.0	1200.0
0.800	0.775	8500.0	5600.0

In their simulation, the algorithms were required to reach a standard rate of accuracy of making no errors in convergence in 100 experiments. The initialization of the reward estimate vector has been done in 20 iterations, which are included in the results shown in the Table 2.4. The results have shown that the continuous TSE algorithm was from 4 to 50% slower than the discrete TSE. For example, with $d_1=0.8$ and $d_2=0.775$, the TSE algorithm took an average of 8500 iterations to converge and the DTSE required only 5600.

Some comparisons were made between all the estimator algorithms, in two benchmark ten-action environments.

Table 2.5: Comparison of the discrete and continuous estimator algorithms in benchmark ten-action environments [10].

Environment	Algorithm	Continuous	Discrete
E _A	Pursuit	1140	799
E _A	TSE	310	207
E _B	Pursuit	2570	1770
E _B	TSE	583	563

Note: The reward probabilities used are:

E_A: 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2

E_B: 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

These environments were the same ones used to compare the continuous estimator algorithms to the L_{RI} scheme. The estimator algorithms sampled all 10 actions, 10 times each, to initialize the estimate vector. These 100 extra iterations are included in the results presented in the Table 2.5. As the simulations of the two-action estimator

algorithms showed, the continuous version of the TSE algorithm is slower than the discrete version; for example, in the environment referred as E_A , the DTSE takes 207 iterations to reach the end-state and the TSE takes 310.

These results also show that the TSE algorithm is faster than the Pursuit Algorithm. In the same environment, the continuous Pursuit Algorithm required 1140 iterations to converge and the TSE algorithm required only 310. The same observation applies to their discrete versions.

2.5 Conclusions

In this chapter, different versions of known learning automata were introduced and presented in a comparative fashion, following their historical development. The main concepts were introduced at the beginning, followed by the description of the Fixed Structure Stochastic Automata. These class of automata was exemplified by presenting the Tsetlin's, Krinsky's and Krylov's automata. The concepts of the Variable Structure Stochastic Automata were described afterward, and different ergodic and absorbing versions of such automata were presented, such as L_{RI} , L_{RP} , L_{IP} . Attention was focused on the probability of converging to the optimal action in different environments, and on the performance of these algorithms.

The discretization process was presented as a subsequent step in the evolution of the learning automata. By discretizing the probability space, and by approaching the optimal solution in discrete steps instead of using continuous functions, the convergence of the discrete learning automata improved considerably. Discrete versions of existing

VSSA were presented, such as DL_{RI} , DL_{RP} , DL_{IP} . The main differences between these discrete algorithms and their continuous versions were explained.

The concepts of the Estimator Algorithms were introduced and exemplified by presenting the Pursuit and the TSE algorithms, with their continuous and discrete versions. Existing experimental results regarding the convergence of these algorithms were presented, and a comparative study was performed based on these results.

This chapter establishes the foundation for the new algorithms and results that will be introduced in the following chapters.

Chapter 3: NEW PURSUIT ALGORITHMS¹

3.1 Introduction

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

Besides these methods of incorporating the acquired knowledge into the probability vector, another important characteristic of the learning automata is represented by the philosophy of the learning paradigm. For example, by giving more importance to rewards than to penalties in the probability updating rules, the learning automata can considerably improve their convergence properties. In the case of the linear schemes of the VSSA, by updating the probability vector $\mathbf{P}(t)$ only if the environment rewarded the chosen action, the linear scheme L_{RI} became ϵ -optimal, whereas the

¹ The work presented in this chapter has been published in "A Comparison of Continuous and Discretized Pursuit Learning Schemes", authored by J. B. Oommen and M. Agache [16].

symmetric linear Reward-Penalty scheme, L_{RP} , is at most expedient. Also, by considerably increasing the value of probability changes on reward in comparison to changes made on penalty, yields a resultant linear scheme, the $L_{R-\epsilon P}$, which is ϵ -optimal. The same behavior can be observed in the case of the FSSA. The difference between the Krinsky automaton and the Tsetlin automaton is that the Krinsky automaton gives more importance to the rewards than to the penalties. This modification improves the performance of the Krinsky automaton, making it ϵ -optimal in all stationary environments, whereas the Tsetlin automaton is ϵ -optimal only in the environments where $\min\{c_1, c_2\} < 0.5$ [31].

In this thesis, we argue that the automaton can model the *long-term* behavior of the Environment by maintaining running estimates of the reward probabilities. Additionally, we contend that the *short-term* perspective of the Environment is also valuable, and we maintain that this information resides in the most recent responses that are obtained by the automaton. We present learning schemes in which both the *short-term* and *long-term* perspectives of the Environment can be incorporated in the learning process. Specifically, the long term information is crystallized in the running reward-probability estimates, and the use of the short term information is achieved by considering whether the most recent response was a reward or a penalty. Thus, when *short-term* perspectives are considered, the Reward-Inaction and the Reward-Penalty learning paradigms become pertinent *in the context of the estimator algorithms*.

The Pursuit algorithm introduced by Thathachar and Sastry [27] considered only the long-term estimates in the probability updating rules. Later, Oommen and Lanctôt

extended the Pursuit Algorithm into the discretized world by presenting the Discretized Pursuit Algorithm, which considered both long-term and short-term estimates, and implemented a reward-inaction learning philosophy [18]. The combination of these learning “philosophies” and paradigms, in conjunction with the continuous and discrete computational models, leads to four versions of Pursuit Learning Automata, listed below:

- i) **Algorithm CP_{RP}**: Continuous Pursuit Reward-Penalty Scheme
Paradigm: Reward-Penalty; Probability Space: Continuous
- ii) **Algorithm CP_{RI}**: Continuous Pursuit Reward-Inaction Scheme
Paradigm: Reward-Inaction; Probability Space: Continuous
- iii) **Algorithm DP_{RP}**: Discretized Pursuit Reward-Penalty Scheme
Paradigm: Reward- Penalty; Probability Space: Discretized
- iv) **Algorithm DP_{RI}**: Discretized Pursuit Reward-Inaction Scheme
Paradigm: Reward-Inaction; Probability Space: Discretized

The CP_{RP} and DP_{RI} algorithms were presented in the previous chapter. This chapter focuses on presenting the Continuous Reward-Inaction Pursuit Algorithm and the Discretized Reward-Penalty Pursuit Algorithm. A comparative study of the performance of the all the Pursuit Algorithms is also presented.

3.2 Continuous Reward-Inaction Pursuit Algorithm (CP_{RI})

The continuous reward-inaction Pursuit Algorithm represents a continuous version of the Discretized Pursuit Algorithm (DP_{RI}) presented by Oommen and Lanctôt [18]. It follows the same learning paradigm as the DP_{RI} algorithm, but in a continuous

probability space. If compared with the CP_{RP} , the CP_{RI} algorithm differs only in the updating probability rules. Being a Reward-Inaction algorithm, it updates the action probability vector $\mathbf{P}(t)$ only if the current action is rewarded by the environment. When the action is penalized, the action probability vector remains unchanged.

A formal description of the algorithm is given as follows:

ALGORITHM CP_{RI}

Parameters

$\lambda, m, e_m, W_i(t), Z_i(t)$: Same as in the CP_{RP} algorithm.

Method

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

Initialize $\hat{d}(t)$ by choosing each action a small number of times.

Repeat

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

Let $\alpha(t) = \alpha_i$.

Step 2: If α_m is the action with the current highest reward estimate, update $\mathbf{P}(t)$ as:

If $\beta(t) = 0$ Then

$$\mathbf{P}(t+1) = (1-\lambda) \mathbf{P}(t) + \lambda e_m \quad (3.1)$$

Else

$$\mathbf{P}(t+1) = \mathbf{P}(t)$$

Step 3: Update $\hat{d}(t)$ exactly as in the CP_{RP} Algorithm

End Repeat

END ALGORITHM CP_{RI}

Similarly to the CP_{RP} , the CP_{RI} algorithm can be proven ϵ -optimal in any stationary environment. The proof for the ϵ -optimality of the CP_{RI} follows the same idea as the other Pursuit algorithms. First, it can be shown that using a sufficiently small value for the learning parameter λ , all actions are chosen enough number of times so that

$\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{d}(t)$ after a finite time.

Mathematically, this can be expressed as follows:

Theorem 3.1: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the CP_{RI} algorithm, for all $\lambda \in (0, \lambda^*)$,

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

Proof:

Let us define the random variable Y_t^i as the number of times the i^{th} action was chosen up to time 't' in any specific realization. From the updating equation (Eq.(3.1)), at any step 't' in the algorithm, we have:

$$p_i(t) \geq p_i(t-1) \cdot (1 - \lambda) \quad (3.2)$$

which implies that during any of the first 't' iterations of the algorithm

$$\Pr\{\alpha_i \text{ is chosen}\} \geq p_i(0) \cdot (1 - \lambda)^t. \quad (3.3)$$

With the above clarified, reminder of the proof is identical to the proof for the CP_{RP} algorithm and omitted [27]. ◆◆◆

It can be shown that if there is an action α_m , for which the reward estimate remains maximal after a finite number of iterations, then the m^{th} component of the action probability vector converges in probability to 1 (see Theorem 3.2).

Theorem 3.2: Suppose that there exists an index m and a time instant $t_0 < \infty$ such that

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

then $p_m(t) \rightarrow 1$ with probability 1 as $t \rightarrow \infty$.

Proof:

To prove this result, we shall demonstrate that the sequence of random variables $\{p_m(t)\}_{t \geq 0}$ is a submartingale. The convergence in probability will result then from the submartingale convergence theorem [11].

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

$$p_m(t+1) = p_m(t) + \lambda \cdot (1 - p_m(t)), \text{ if } \beta(t)=0 \text{ (i.e., with probability } d_m) \quad (3.4)$$

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

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

Then, the quantity:

$$\Delta p_m(t) = E[p_m(t+1) - p_m(t) \mid Q(t)] \quad (3.5)$$

becomes

$$\Delta p_m(t) = d_m \cdot \lambda \cdot (1 - p_m(t)) \geq 0, \text{ for all } t \geq t_0, \quad (3.6)$$

which implies that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [11], $\{p_m(t)\}_{t \geq 0}$ converges as $t \rightarrow \infty$,

$$E[p_m(t+1) - p_m(t) \mid Q(t)] \rightarrow 0 \text{ with probability } 1. \quad (3.7)$$

Hence, $p_m(t) \rightarrow 1$ with probability 1, and the theorem is proven. $\blacklozenge \blacklozenge \blacklozenge$

Finally, the Theorem 3.3 expresses the ε -optimality convergence for the CP_{RI} algorithm, and can be easily deducted from the two previous results.

Theorem 3.3: For the CP_{RI} algorithm, in every stationary random environment, there exists a $\lambda^* > 0$ and $t_0 > 0$, such that for all $\lambda \in (0, \lambda^*)$ and for any $\delta \in (0, 1)$ and any $\varepsilon \in (0, 1)$,

$$\Pr[p_m(t) > 1 - \varepsilon] > 1 - \delta, \text{ for all } t > t_0.$$

In order to study the performance of this algorithm, simulations and comparisons against different estimator algorithms were performed. The results are presented in the Section 3.4.

3.3 Discretized reward-penalty Pursuit Algorithm (DP_{RP})

A new Pursuit algorithm, denoted DP_{RP}, is obtained by combining the strategy of discretizing the probability space with the Reward-Penalty learning paradigm in the context of a Pursuit “philosophy”. This algorithm uses the estimates in updating the action probability vector $\mathbf{P}(t)$ if the environment penalizes or rewards the chosen action. As any discrete algorithm, the DP_{RP} algorithm performs changes to the action probability vector $\mathbf{P}(t)$ in a discrete fashion. Each component of $\mathbf{P}(t)$ can change with a value which is a multiple of Δ , where $\Delta=1/rN$, r is the number of actions and N is a resolution parameter. Formally, the algorithm can be expressed as follows:

ALGORITHM DP_{RP}

Parameters

m , $W_i(t)$, $Z_i(t)$, N and Δ : Same as in the CP_{RP} algorithm.

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by choosing each action a small number of times.

Repeat

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

Let $\alpha(t) = \alpha_i$.

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

If $\beta(t)=0$ and $p_m(t) \neq 1$ Then

$$\begin{aligned}
p_j(t+1) &= \max_{j \neq m} \{p_j(t) - \Delta, 0\} \\
p_m(t+1) &= 1 - \sum_{j \neq m} p_j(t+1)
\end{aligned} \tag{3.8}$$

Step 3: Update $\hat{d}(t)$ exactly as in the CP_{RP} Algorithm

End Repeat

END ALGORITHM DP_{RP}

Similarly to the DP_{RI} algorithm, it is possible to show that DP_{RP} algorithm is ε -optimal in every stationary environment. The proof is very similar to the proof for convergence of the DP_{RI} and it follows the same steps. First, we can prove that if the m^{th} action is rewarded more than any other action from time t_0 onward then the action probability vector $\mathbf{P}(t)$ for the DP_{RP} will converge to the unit vector $\mathbf{e}_m$. The next theorem captures this result:

Theorem 3.4: If there exists an index m and a time instant $t_0 < \infty$ such that $\hat{d}_m(t) > \hat{d}_j(t)$ for all j , $j \neq m$ and all $t \geq t_0$; then there exists an integer N_0 such that for all resolution parameters $N > N_0$, $p_m(t) \rightarrow 1$ with probability 1 as $t \rightarrow \infty$.

Proof: The proof for this theorem aims to show that $\{p_m(t)\}_{t \geq t_0}$ is a submartingale satisfying $\sup_{t \geq 0} E[|p_m(t)|] < \infty$. Then, based on the submartingale convergence theorem

[11] $\{p_m(t)\}_{t \geq t_0}$ converges, which implies

$$E[p_m(t+1) - p_m(t) | Q(t)] \xrightarrow{t \rightarrow \infty} 0. \tag{3.9}$$

From the probability updating equations, results

$$E[p_m(t+1) | Q(t), p_m(t) \neq 1] = p_m(t) + d_m c_i \Delta, \tag{3.10}$$

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

$$E[p_m(t+1) - p_m(t) | Q(t)] = d_m c_t \Delta \geq 0, \text{ for all } t \geq t_0, \quad (3.11)$$

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

◆◆◆

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

Theorem 3.5: If for each action α_i , $p_i(0) \neq 0$, then for any given constants $\delta > 0$ and $M < \infty$, there exists $N_0 < \infty$ and $t_0 < \infty$ such that, under DP_{RP} , 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: The proof for this theorem is identical with the proof for the same theorem in the case of the DP_{RI} algorithm, published in [18].

◆◆◆

These two theorems lead to the conclusion that the DP_{RP} scheme is ϵ -optimal in all stationary random environments.

3.4 Simulation Results

In the context of this study, simulations were performed in order to compare the rates of convergence of different Pursuit Algorithms in benchmark environments. In all the tests performed, an algorithm was considered to have converged if the probability of choosing an action exceeded a threshold T ($0 < T \leq 1$). If the automaton converged to the best action (*i.e.*, the one with the highest probability of being rewarded), it was considered to have converged correctly.

Before comparing the performance of the automata, innumerable multiple tests were executed to determine the “best” value of the respective learning parameters for each individual algorithm. The value was reckoned as the “best” value if it yielded the fastest convergence and the automaton converged to the correct action in a sequence of NE experiments. These best parameters were then chosen as the final parameter values used for the respective algorithms to compare their rates of convergence.

When the simulations were performed considering the same threshold T and number of experiments, NE, as Oommen and Lanctôt did in [18], (*i.e.* $T=0.99$ and $NE=75$), the learning parameters obtained for the DP_{RI} algorithm in the (0.8 0.6) environment had a variance coefficient² of 0.2756665 in 40 tests performed. This variance coefficient was not considered satisfactory for comparing the performance of the Pursuit algorithms. Subsequent simulations were performed imposing stricter convergence requirements by increasing the threshold T , and proportionally, the number

² The variance coefficient is defined as σ/M , where σ is the standard deviation and M is the mean.

of experiments NE, which yielded learning parameters with smaller variance coefficients. For example, the learning parameter (N) for DP_{RI} algorithm in the (0.8 0.6) environment, when $T=0.999$ and $NE=750$, exhibits a variance coefficient of 0.0706893, which represents a much smaller variance. Therefore, in this thesis the simulation results for $T=0.999$ and NE equal to 500 and 750 experiments shall be presented.

The simulations were performed in different existing benchmark environments with two and ten actions. These environments have also been used to compare a variety of continuous and discretized schemes, and in particular the DP_{RI} in [18] and to compare the performance of the CP_{RP} against other traditional VSSA in [27]. Furthermore, to keep conditions identical, every estimator algorithm sampled all actions 10 times each to initialize the estimate vector. These extra iterations are also included in the results presented in the following tables. Table 3.1 and Table 3.2 contain the simulation results for these four algorithms in two action environments. The probability of reward for one action was fixed at 0.8 for all simulations and the probability of reward for the second action was increased from 0.5 to 0.7³. In each case, the reported results correspond to the results obtained using the above-described “best” parameters.

³ The reader will observe that there is a considerable difference between the results presented here and the results presented in [29]. In [29], the parameter chosen was the one which gave correct convergence in 25 parallel experiments. However, on testing the CP_{RP} for 1000 experiments, it was observed that it yielded only 84% accuracy. Thus, in the case of the CP_{RP} what we seek is the largest parameter, λ , which yields correct convergence *in all the 750 and 500 experiments respectively*. Similarly, in the case of the DP_{RP} we seek the smallest integer parameter, N, which yields correct convergence *in all the 750 and 500 experiments respectively*.

Table 3.1: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$).

Environ.		DP _{RI}		DP _{RP}		CP _{RI}		CP _{RP}	
d ₁	d ₂	N	No. of Iterat.	N	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.
0.8	0.5	20	49.07	32	53.74	0.214	55.45	0.122	69.69
0.8	0.6	58	105.64	89	118.24	0.046	198.32	0.027	258.60
0.8	0.7	274	430.20	391	456.13	0.009	939.63	0.0072	942.88

Table 3.2: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 500 experiments ($NE=500$).

Environ.		DP _{RI}		DP _{RP}		CP _{RI}		CP _{RP}	
d ₁	d ₂	N	No. of Iterat.	N	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.
0.8	0.5	17	44.75	26	47.9	0.314	43.12	0.169	55.50
0.8	0.6	52	97	74	102.17	0.054	171.85	0.036	199.93
0.8	0.7	217	357.99	297	364.09	0.011	789.29	0.0075	905.36

The results of these simulations suggest that, as the difference in the reward probabilities decreases, *i.e.* as the environment gets more difficult to learn, the **Discretized Pursuit** algorithms exhibit a performance superior to the **Continuous** algorithms. Also, comparing the Pursuit algorithms based on the Reward-Inaction paradigm with the Pursuit algorithms based on the Reward-Penalty paradigm, one can notice that, in general, the Pursuit Reward-Inaction algorithms are up to 20% faster than the Reward-Penalty Pursuit algorithms. For example, if $d_1=0.8$ and $d_2=0.6$, the discretized DP_{RI} converges to the correct action in an average of 105.64 iterations, and

the DP_{RP} algorithm converges in an average of 118.24 iterations. In the same environment, the CP_{RI} algorithm takes an average of 198.32 iterations and the CP_{RP} requires 258.60, indicating that the CP_{RI} is 23% faster than the CP_{RP} algorithm.

Figure 3.1 illustrates the performance of these algorithms relative to the performance of the CP_{RP} algorithm.

Figure 3.1: Comparison of the Pursuit algorithms in two-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$).

The existing two-action benchmark environments used to obtain the results presented in Table 3.1 and Table 3.2 are characterized by having high reward probabilities. To accurately reflect the performance of these algorithms, we have performed the same simulations in three other environments in which the probability of reward are small. For this purpose, we fixed the reward probability of the first action to

0.2 and we varied the reward probability of the second action from 0.5 to 0.3. Table 3.3 presents the results obtained in these environments.

Table 3.3: Comparison of the Pursuit algorithms in new two-action environments for which exact convergence was required in 750 experiments ($NE=750$).

Environ.		DP _{RI}		DP _{RP}		CP _{RI}		CP _{RP}	
d_1	d_2	N	No. of Iterat.	N	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.
0.2	0.5	12	51.29	38	60.16	0.349	54.25	0.089	89.25
0.2	0.4	27	108.12	100	129.69	0.109	172.14	0.0273	255.01
0.2	0.3	89	403.59	402	479.69	0.0294	797.78	0.005	1321.11

The results show that also in these environments, the schemes that employed the Reward-Inaction learning paradigm exhibit higher performance than the Reward-Penalty schemes. Furthermore, the discretized schemes prove to be faster than the continuous schemes.

For completeness, similar experiments were performed in the benchmark ten-action environments [18], [TO]. Table 3.4 and Table 3.5 present the results obtained in these environments for 750 and 500 experiments.

Table 3.4: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$).

Envi ron.	DP_{RI}		DP_{RP}		CP_{RI}		CP_{RP}	
	N	No. of Iterat.	N	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.
E_A	188	752.7	572	1126.8	0.0097	1230.3	0.003	2427.3
E_B	1060	2693.7	1655	3230.3	0.002	4603	0.00126	5685

Note: The Reward probabilities for the actions are:

E_A : 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2

E_B : 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

Table 3.5: Comparison of the Pursuit algorithms in ten-action benchmark environments for which exact convergence was required in 500 experiments ($NE=500$).

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

As in the previous two-action environments, in ten-action environments, the DP_{RI} algorithm proved to have the best performance, converging to the correct action almost 25% faster than the DP_{RP} algorithm, and almost 50% faster than the CP_{RP} algorithm. If we analyze the behavior of these automata in the first environment, E_A , when $NE=750$, the average number of iterations required by the DP_{RI} to converge is 752.7, whereas the DP_{RP} required 1126.8, implying that the DP_{RI} algorithm is 33% faster than the DP_{RP} algorithm. In the same environment, the CP_{RI} requires an average of 1230.3 iterations for

convergence, and the CP_{RP} requires 2427.3, which shows that the CP_{RI} algorithm is 50% faster than the CP_{RP} algorithm, and the DP_{RI} is almost 70% faster than the CP_{RP} .

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

Best Algorithm: Discretized Pursuit Reward-Inaction (DP_{RI})

2nd-best Algorithm: Discretized Pursuit Reward-Penalty (DP_{RP})

3rd-best Algorithm: Continuous Pursuit Reward-Inaction (CP_{RI})

4th-best Algorithm: Continuous Pursuit Reward-Penalty (CP_{RP})

Also, the simulation results have shown that the discretized Pursuit algorithms are up to 30% faster than their continuous counterparts. Furthermore, comparing the Reward-Inaction Pursuit algorithms against the Reward-Penalty algorithms it can be seen that the Reward-Inaction algorithms are superior in the rate of convergence; they are up to 25% faster than their Reward-Penalty counterparts.

3.5 Conclusions

This chapter extends the class of Pursuit Algorithms by introducing two new algorithms resulted from the combination of the Reward-Penalty and Reward-Inaction learning paradigms in conjunction with the continuous and discrete models of computation. The new algorithms introduced are the Continuous Reward-Inaction Pursuit Algorithm and the Discretized Reward-Penalty Pursuit algorithm. Furthermore, in this chapter we argue that a learning scheme that utilizes the most recent response of

the Environment permits the learning algorithm to utilize the *long-term* and *short-term* perspectives of the Environment.

This chapter contains a detailed description of these algorithms and the proofs of their convergence. Also, simulation results regarding the convergence of these algorithms were performed and a quantitative comparison between the performance of all the Pursuit algorithms was presented.

Overall, the Discrete Pursuit Reward-Inaction algorithm surpasses the performance of all the other versions of Pursuit algorithms. Also, the Reward-Inaction schemes are superior to their Reward-Penalty counterparts.

Chapter 4: GENERALIZATION OF THE PURSUIT ALGORITHM⁴

4.1 Introduction

The main idea that characterizes the Pursuit algorithms presented in the previous chapters is that they '*pursue*' the best-estimated action, which is the action corresponding to the maximal estimate $\hat{d}_{\max}(t)$. In any iteration, these algorithms increase only the probability of the best action, ensuring that the probability vector $\mathbf{P}(t)$ moves towards the solution that has the maximal estimate at the current time. This implies that if, at any time 't', the action that has the maximum estimate is not the action that has the minimum penalty probability, then the automaton pursues a wrong action.

In an attempt to minimize this probability of pursuing a wrong action, our goal in this chapter is to generalize the design of the Pursuit algorithm such that it pursues a set of actions. Specifically, these actions have higher reward estimates than the current chosen action.

⁴ The work presented in this chapter is comprised in "*Continuous and Discretized Generalized Pursuit Learning Schemes*", authored by M. Agache and J.B. Oommen, submitted for publication at SCT'2000 [1].

Figure 4.1 presents a pictorial representation of the two Pursuit approaches of converging to an action. The first approach, adopted by the Pursuit Algorithms described in Chapter 3, such as CP_{RP} , CP_{RI} , DP_{RP} , DP_{RI} , always pursues the best-estimated action. The present approach, adopted by the Generalized Pursuit Algorithms which we present here, does not follow only the best action - it follows all the actions that are “better” than the current chosen action, *i.e.* the actions that have higher reward estimates than the chosen action.

Figure 4.1: Solution approach of the CP_{RP} Pursuit Algorithm and the Generalized Pursuit algorithms

In a vectorial form, if action α_m is the action that has the highest reward estimate at time ‘t’, the Pursuit Algorithms always pursue the vector $e(t) = [0 \ 0 \ \dots \ 1 \ 0 \dots 0]^T$, where $e_m(t)=1$. In contrast, if α_i denotes the chosen action, the Generalized Pursuit Algorithm pursues the vector $e(t)$, where

$$\begin{aligned}
 e_j(t) &= \begin{cases} 0, & \text{if } \hat{d}_j(t) \leq \hat{d}_i(t) \\ 1, & \text{if } \hat{d}_j(t) > \hat{d}_i(t) \end{cases} \quad \text{for } j \neq i \\
 e_i(t) &= \begin{cases} 1, & \text{if } \hat{d}_i(t) = \max\{\hat{d}_j(t)\} \\ 0, & \text{otherwise} \end{cases}
 \end{aligned} \tag{4.1}$$

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

This chapter presents two versions of Generalized Pursuit algorithms, followed by a comparative study of the performance of these algorithms with the existing Pursuit algorithms. The first algorithm introduced in this chapter is the Generalized Pursuit Algorithm. This algorithm moves the action probability vector “away” from the actions that have smaller reward estimates, but it does not guarantee that it increases the probability for all the actions with higher estimates than the chosen action.

Next, a discretized Generalized Pursuit Algorithm is presented. This algorithm follows the philosophy of a Generalized Pursuit Algorithm in the sense that it increases the action probability for all the actions with higher reward estimates than the current chosen action.

Due to their generalized philosophy, in environments with two actions, these algorithms degenerate to become the existing Pursuit algorithms. Specifically, in two action environments, the GPA algorithm becomes the CP_{RP} algorithm, and the DGPA algorithm becomes the DP_{RP} algorithm. For this reason, these algorithms were tested only in benchmark ten-action environments, and the simulation results are presented in the section 4.4 of this chapter.

The following sections present the new Generalized Pursuit Algorithms.

4.2 Generalized Pursuit Algorithm

The Generalized Pursuit Algorithm (GPA) presented in this section, is an example of an algorithm that generalizes the Pursuit Algorithm introduced by Thathachar and Sastry in [28], referred to as the CP_{RP} algorithm. It is a continuous estimator algorithm, which moves the probability vector towards a set of possible solutions in the probability space. Each possible solution is a unit vector in which the value '1' corresponds to an action that has a higher reward estimate than the chosen action.

The CP_{RP} algorithm increases the probability for the action that has the higher reward estimate, and decreases the action probability for all the other actions, as shown in the following updating equations:

$$\begin{aligned} p_m(t+1) &= (1-\lambda)p_m(t) + \lambda, & \text{where } \hat{d}_m &= \max_{j=1, \dots, r} \{\hat{d}_j(t)\} \\ p_j(t+1) &= (1-\lambda)p_j(t), & \text{for all } j &\neq m \end{aligned} \quad (4.2)$$

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

$$p_j(t+1) = (1-\lambda)p_j(t), \quad j = 1, \dots, r$$

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

$$\Delta = 1 - \sum_{j=1}^r p_j(t+1) = 1 - \sum_{j=1}^r (1-\lambda)p_j(t) = 1 - \sum_{j=1}^r p_j(t) + \lambda \sum_{j=1}^r p_j(t) = \lambda$$

In order to increase the probability of the best-estimated action, the CP_{RP} Pursuit algorithm adds the probability mass Δ to the probability of the best-estimated action:

$$p_m(t+1) = (1-\lambda)p_m(t) + \Delta = (1-\lambda)p_m(t) + \lambda, \quad \text{where } \hat{d}_m = \max_{j=1, \dots, r} \{\hat{d}_j(t)\}$$

In contrast to the CP_{RP} algorithm, the newly introduced GPA algorithm equally distributes the remaining amount Δ to all the actions that have higher estimates than the chosen action. If $K(t)$ denotes the number of actions that have higher estimates than the chosen action at time 't', then the updating equations for the Generalized Pursuit Algorithm are expressed by the following equations:

$$\begin{aligned} p_j(t+1) &= (1-\lambda)p_j(t) + \frac{\lambda}{K(t)}, & \text{for all } j \neq i, \text{ such that } \hat{d}_j(t) > \hat{d}_i(t) \\ p_j(t+1) &= (1-\lambda)p_j(t), & \text{for all } j \neq i, \text{ such that } \hat{d}_j(t) \leq \hat{d}_i(t) \\ p_i(t+1) &= 1 - \sum_{j \neq i} p_j(t). \end{aligned} \quad (4.3)$$

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

$$\mathbf{P}(t+1) = (1-\lambda) \cdot \mathbf{P}(t) + \frac{\lambda}{K(t)} \cdot \mathbf{e}(t), \quad (4.4)$$

where $\mathbf{e}(t)$ is the *direction vector* defined in (Eq 4.1).

Based on these equations, it can be seen that the GPA algorithm increases the probability for all the actions with higher reward estimates than the estimate of the chosen action, and satisfying the following inequality:

$$p_j(t) < \frac{1}{K(t)} \quad (4.5)$$

Formally, the Generalized Pursuit Algorithm can be described as follows:

ALGORITHM GPA

Parameters

λ the learning parameter , where $0 < \lambda < 1$

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

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

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

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking each action a small number of times.

Repeat

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

Step 2: If $K(t)$ represents the number of actions with higher estimates than the chosen action at time t , update $\mathbf{P}(t)$ according to the following equations:

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

$$p_j(t+1) = p_j(t) - \lambda \cdot p_j(t) \quad (\forall j) \text{ such that } \hat{d}_j(t) \leq \hat{d}_i(t) \quad (4.6)$$

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

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

$$W_i(t+1) = W_i(t) + (1 - \beta(t))$$

$$Z_i(t+1) = Z_i(t) + 1 \quad (4.7)$$

$$\hat{d}_i(t+1) = \frac{W_i(t+1)}{Z_i(t+1)}$$

$$\left. \begin{array}{l} W_j(t+1) = W_j(t) \\ Z_j(t+1) = Z_j(t) \\ \hat{d}_j(t+1) = \hat{d}_j(t) \end{array} \right\} , \text{ for all } j \neq i \quad (4.8)$$

End Repeat

END ALGORITHM GPA

As for the previous Pursuit algorithms, the convergence of the GPA is proven in two steps. First, we demonstrate that using a sufficiently small value for the learning

parameter λ , all actions are chosen enough number of times such that $\hat{d}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. Formally, this is expressed as follows:

Theorem 4.1: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the GPA algorithm, for all $\lambda \in (0, \lambda^*)$,

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

Proof: The proof of this theorem is analogous to the proof of the corresponding result for the TSE algorithm. We shall consider the same random variable Y_t^i as the number of times the i^{th} action was chosen up to time 't' in any specific realization.

From the updating equation (Eq.(4.4)), at any step 't' in the algorithm, if the action α_i is chosen, we have:

$$\begin{aligned} p_j(t) &= (1 - \lambda) \cdot p_j(t-1) + \frac{\lambda}{K(t)} && \text{if } \hat{d}_j(t-1) > \hat{d}_i(t), j \neq i \\ p_j(t) &= (1 - \lambda) \cdot p_j(t-1) && \text{if } \hat{d}_j(t-1) \leq \hat{d}_i(t-1), j \neq i \end{aligned} \quad (4.9)$$

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

$$p_i(t) \geq (1 - \lambda) \cdot p_i(t-1) \quad (4.10)$$

The equations (4.9) and (4.10) show that the following inequality is valid for all the actions:

$$p_j(t) \geq (1 - \lambda) \cdot p_j(t-1), \quad (\forall j) j = 1, \dots, r \quad (4.11)$$

which implies that during any of the first 't' iterations of the algorithm

$$\Pr\{\alpha_i \text{ is chosen}\} \geq p_i(0) \cdot (1 - \lambda)^t, \text{ for any } i=1, \dots, r \quad (4.12)$$

With the above clarified, the reminder of the proof is identical to the proof for the TSE algorithm and omitted (it can be found in [28]). ◆◆◆

The second step in proving the convergence of the GPA consists of demonstrating that if the m^{th} action is rewarded more than any other action from time t_0 onward, then the action probability vector converges in probability to e_m . This is shown in Theorem 4.2.

Theorem 4.2: Suppose that there exists an index m and a time instant $t_0 < \infty$ such that

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

then $p_m(t) \rightarrow 1$ with probability 1 as $t \rightarrow \infty$.

Proof: We shall demonstrate that the sequence of random variables $\{p_m(t)\}_{t \geq 0}$ is a submartingale. The convergence in probability results then from the submartingale convergence theorem [11].

Consider

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

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

From the updating equations (4.4) and the assumptions of this theorem, $p_m(t+1)$ can have be expressed as:

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

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

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

$$\begin{aligned} \Delta p_m(t) &= \sum_{j \neq m} \left(\frac{\lambda}{K(t)} - \lambda \cdot p_m(t) \right) \cdot p_j(t) + [\lambda(1 - p_m(t))] \cdot p_m(t) = \\ &= \lambda \left(\frac{1}{K(t)} - p_m(t) \right) \cdot (1 - p_m(t)) + \lambda(1 - p_m(t)) \cdot p_m(t) = \\ &= \frac{\lambda}{K(t)} (1 - p_m(t)) \geq 0 \end{aligned} \quad (4.13)$$

Hence, $p_m(t)$ is a submartingale. By the submartingale convergence theorem [11],

$\{p_m(t)\}_{t \geq t_0}$ converges as $t \rightarrow \infty$,

$E[p_m(t+1) - p_m(t) \mid Q(t)] \rightarrow 0$ with probability 1.

Hence, $p_m(t) \rightarrow 1$ with probability 1, and the theorem is proven. ◆◆◆

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

Theorem 4.3: For the GPA algorithm, in every stationary random environment, there exists a $\lambda^* > 0$ and $t_0 > 0$, such that for all $\lambda \in (0, \lambda^*)$ and for any $\delta \in (0, 1)$ and any $\varepsilon \in (0, 1)$,

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

for all $t > t_0$. ◆◆◆

The proof for this theorem results as a logical consequence of the previous two theorems, Theorem 4.1 and Theorem 4.2.

The simulation results regarding the performance of this algorithm are presented in the section 4.4 of this chapter. The next section presents another version of a Generalized Pursuit algorithm.

4.3 Discretized Generalized Pursuit Algorithm

The Discretized Generalized Pursuit Algorithm, denoted DGPA, is another algorithm that generalizes the concepts of the Pursuit algorithm by “pursuing” all the actions that have higher estimates than the current chosen action. This algorithm moves the probability vector $P(t)$ in discrete steps, but the steps do not have equal sizes.

At each iteration, the algorithm counts how many actions have higher estimates than the current chosen action. If $K(t)$ denotes this number, the DGPA algorithm increases the probability of all the actions with higher estimates with the amount $\Delta/K(t)$, and decreases the probabilities for all the other actions with the amount $\Delta/(r-K(t))$, where Δ is a resolution step, $\Delta=1/rN$ with N a resolution parameter.

Vectorially, the updating equations can be expressed as follows:

$$P(t+1) = P(t) + \frac{\Delta}{K(t)} \cdot e(t) - \frac{\Delta}{r-K(t)} \cdot [u - e(t)] \quad (4.14)$$

where $e(t)$ is the direction vector defined in (Eq. 4.1) and u is the unit vector $u_j=1$, $j=1,2,\dots,r$.

A detailed description of the algorithm is given below:

ALGORITHM DGPA**Parameters****N** resolution parameter**K(t)** the number of actions with higher estimates than the current chosen action **Δ** the smallest step size $\Delta = \frac{1}{rN}$ **$W_i(t)$** the number of times the i^{th} action has been rewarded up to the time t , with $1 \leq i \leq r$ **$Z_i(t)$** the number of times the i^{th} action has been chosen up to the time t , with $1 \leq i \leq r$ **Method****Initialization** $p_i(t) = 1/r$, for $1 \leq i \leq r$ Initialize $\hat{d}(t)$ by picking each action a small number of times.**Repeat**Step 1: At time t pick $\alpha(t)$ according to probability distribution $P(t)$. Let $\alpha(t) = \alpha_i$.Step 2: Update $P(t)$ according to the following equations:

$$p_j(t+1) = \min\left\{p_j(t) + \frac{\Delta}{K(t)}, 1\right\} \quad (\forall j) j \neq i, \text{ such that } \hat{d}_j(t) > \hat{d}_i(t)$$

$$p_j(t+1) = \max\left\{p_j(t) - \frac{\Delta}{r - K(t)}, 0\right\} \quad (\forall j) j \neq i, \text{ such that } \hat{d}_j(t) \leq \hat{d}_i(t) \quad (4.15)$$

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

Step 3: Same as in the GPA algorithm

End Repeat**END ALGORITHM DGPA**

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

To prove the convergence of this algorithm, we shall first prove that the DGPA possesses the moderation property (see section 2.5.3). Then we prove that this algorithm also possesses the monotone property. As a consequence of these properties, the Markov chain $\{p_m(t)\}$ is a submartingale [10], and by the submartingale convergence theorem [11], results the convergence of the algorithm.

Theorem 4.4: The DGPA possesses the moderation property.

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

$$p_j(t) - p_j(t+1) < \frac{1}{rN} \quad (4.16)$$

From the updating equations (4.15), the amount that a probability decreases is computed to:

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

and the result is proven. ◆◆◆

The second step in proving the convergence of the DGPA consists of demonstrating that the DGPA possesses the monotone property. This is shown in Theorem 4.5.

Theorem 4.5: The DGPA possesses the monotone property.

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

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

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

Proof: The proof for this theorem aims to show that $\{p_m(t)\}_{t \geq t_0}$ is a submartingale satisfying $\sup_{t \geq 0} E[p_m(t)] < \infty$. Then, based on the submartingale convergence theorem

[11] $\{p_m(t)\}_{t \geq t_0}$ converges, and so,

$$E[p_m(t+1) - p_m(t) | Q(t)] \xrightarrow{t \rightarrow \infty} 0.$$

Consider

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

where $Q(t)$ is the state vector for the estimator algorithms.

From the updating equations (4.15) and the assumptions of this theorem, $p_m(t+1)$ can be expressed as:

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

$$p_m(t+1) = 1 - \sum_{j \neq m} \left(p_j(t) - \frac{\Delta}{r-1} \right) = p_m(t) + \Delta \quad \text{if } \alpha_m \text{ is chosen}$$

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

$$\begin{aligned} E[p_m(t+1) - p_m(t) | Q(t)] &= \sum_{j \neq m} \frac{\Delta}{K(t)} \cdot p_j(t) + \Delta \cdot p_m(t) \\ &= \frac{\Delta}{K(t)} (1 - p_m(t)) + \Delta \cdot p_m(t) \\ &= \frac{\Delta}{K(t)} + \Delta \cdot p_m(t) \left(1 - \frac{1}{K(t)} \right) > 0 \end{aligned} \quad (4.17)$$

Hence, $p_m(t)$ is a submartingale. By the submartingale convergence theorem [11],

$\{p_m(t)\}_{t \geq t_0}$ converges as $t \rightarrow \infty$,

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

Hence, $p_m(t) \rightarrow 1$ with probability 1, and the theorem is proven. ◆◆◆

Since the DGPA possesses the moderation and monotony properties, it implies that the DGPA is ε -optimal [10].

The next section presents the simulation results for the Generalized Pursuit algorithms introduced in this chapter.

4.4 Simulation Results

The simulation results presented in this section have been obtained in the same conditions as the results presented in section 3.4. First, simulations were performed in order to determine the optimal learning parameter for each learning algorithm. Then, using these learning parameters, each algorithm was required to converge in 750 experiments and the average number of iterations was computed. For each algorithm, 100 test were performed and the number of iterations presented in this section is the average over these experiments.

In a two-action environment, the GPA algorithm reduces in a degenerate manner, to the CP_{RP} Pursuit algorithm, and the DGPA reduces to DP_{RP} algorithm. For this reason, simulations were performed only in ten-action benchmark environments and the results are presented in Table 4.1. For simulation results of the CP_{RP} and DP_{RP} algorithms, see section 3.4.

Table 4.1: Performance of the generalized Pursuit algorithms in benchmark ten-action environments for which exact convergence was required in 750 experiments ($NE=750$)

Environ- ment	GPA		DGPA		CP _{RP}		DP _{RI}	
	λ	No. of Iterat.	N	No. of Iterat.	λ	No. of Iterat.	N	No. of Iterat.
E _A	0.0127	948.03	24	633.64	0.003	2427.3	188	752.7
E _B	0.0041	2759.02	52	1307.76	0.00126	5685	1060	2693.7

Note: The reward probabilities for the actions are:

E_A: 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2

E_B: 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

Since the GPA algorithm was designed as a generalization of the continuous reward-penalty Pursuit algorithm, a comparison between these two algorithms is presented. The results show that the GPA algorithm is 60% faster than the CP_{RP} algorithm in the E_A environment and 51% faster in the E_B environment. For example, in the E_A environment, the GPA converges in average in 948.03 iterations, and the CP_{RP} algorithm requires in average 2427 iterations for convergence, which shows an improvement of 60%. In the E_B environment, the CP_{RP} algorithm required on average 5685 number of iterations whereas the GPA algorithm required on average only 2759.02 iterations for convergence, being 51% faster than the CP_{RP} algorithm.

Similarly, the DGPA is classified as a reward-penalty discretized Pursuit algorithm. If compared against the DP_{RP} algorithm, the DGPA proves to be up to 59% faster. For example, in the E_B environment, the DGPA algorithm converges in an average of 1307.76 iterations whereas the DP_{RP} algorithm requires 3230 iterations. Also,

the DGPA algorithm proves to be the fastest Pursuit algorithm, being up to 50% faster than the DP_{RI} algorithm. For example, in the same environment E_B , the DGPA algorithm requires 1307.76 and the DP_{RI} algorithm requires 2693 iterations for convergence.

The following figure presents the graphical representation of the relative performance of these algorithms to the CP_{RP} algorithm in benchmark ten-action environments.

Figure 4.2: Performance of the Pursuit Algorithms relative to the CP_{RP} algorithm in ten-action environments for or which exact convergence was required in 750 experiments

Based on these experimental results and considering the number of iterations required to attain the same accuracy of convergence, we can rank the six Pursuit algorithms as follows:

Best Algorithm: Discretized Generalized Pursuit Algorithm (DGPA)

2nd-best Algorithm: Discretized Pursuit Reward-Inaction (DP_{RI})

3rd-best Algorithm: Generalized Pursuit Algorithm (GPA)

4rd-best Algorithm: Discretized Pursuit Reward-Penalty (DP_{RP})

5rd-best Algorithm: Continuous Pursuit Reward-Inaction (CP_{RI})

6th-best Algorithm: Continuous Pursuit Reward-Penalty (CP_{RP})

4.5 Conclusions

This chapter introduced another extension of the class of Pursuit estimator algorithms by presenting the generalization concept of the Pursuit algorithm. The Pursuit algorithms presented in Chapters 3 “pursue” the action that has the maximal reward estimate at any iteration. Through generalization, the generalized Pursuit algorithms “pursue” the actions that have higher estimates than the current chosen action, minimizing the probability of pursuing a wrong action.

Two versions of generalized Pursuit algorithms were presented in this chapter, the Generalized Pursuit Algorithm (GPA) and the Discretized Generalized Pursuit Algorithm (DGPA), along with the proofs for their convergence.

Simulation results were presented to characterize the performance of these algorithms. Based on the experimental results, the generalized Pursuit algorithms prove to be faster than Pursuit algorithms in environments with more than two actions. Furthermore, in the same environments, the Discretized Generalized Pursuit Algorithm proves to be the fastest converging Pursuit algorithm.

Chapter 5: GENERALIZATION OF THE TSE ALGORITHM⁵

Thathachar and Sastry introduced the concept of estimator learning algorithms by presenting the TSE algorithm [28]. As described in Section 2.5.2.2, the authors of this algorithm presented its updating equations in a scalar form. In this chapter, we first present a vectorial representation of the updating equations of the TSE algorithm. A generalization of the TSE algorithm, derived from its vectorial form, is presented next, along with simulation results that characterize the performance of this algorithm.

5.1 Vectorial representation of the TSE algorithm

The purpose of this section is to present a vectorial representation of the TSE algorithm, which better outlines the underlying concepts of this algorithm.

The TSE algorithm is an estimator algorithm that moves the probability vector $\mathbf{P}(t)$ towards the unit vectors associated with all the actions α_j that have higher reward estimates than the chosen action α_i

$$\hat{d}_j(t) > \hat{d}_i(t).$$

⁵ The contributions presented in this chapter are currently being compiled in a paper that will be submitted for publication.

Viewed from a scalar perspective, the updating equations can be specified as in (2.39), where these actions are identified using the indicator function $S_{ij}(t)$. To identify the same set of actions in vectorial form, a vector *direction* $\mathbf{e}(t)$ can be defined as follows:

$$\mathbf{e}(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \\ \dots \\ e_{r-1}(t) \\ e_r(t) \end{bmatrix}, \text{ where}$$

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

$$e_i(t) = \begin{cases} 1, & \text{if } \hat{d}_i(t) = \max \{ \hat{d}_j(t) \} \\ 0, & \text{otherwise} \end{cases} \quad (5.1)$$

The actions α_k for which $e_k(t)=1$ have the action probability $p_k(t+1)$ increased, whereas the actions for which $e_k(t)=0$ have the action probability $p_k(t+1)$ decreased.

From the updating equations (2.39), it can be seen that the TSE algorithm increases the action probability proportional to the distance to the estimate of the chosen action $\hat{d}_j(t) - \hat{d}_i(t)$. To represent the same concept in a vectorial form, we define a *distance* matrix $\mathbf{F}_{r \times r}$ as follows:

$$\mathbf{F}_i(t) = \begin{bmatrix} f(|\hat{d}_i(t) - \hat{d}_1(t)|) & 0 & \dots & 0 & \dots & 0 \\ 0 & f(|\hat{d}_i(t) - \hat{d}_2(t)|) & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ -f(|\hat{d}_i(t) - \hat{d}_1(t)|) & -f(|\hat{d}_i(t) - \hat{d}_2(t)|) & \dots & 0 & \dots & -f(|\hat{d}_i(t) - \hat{d}_r(t)|) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & f(|\hat{d}_i(t) - \hat{d}_r(t)|) \end{bmatrix} \quad (5.2)$$

where $f: [-1,1] \rightarrow [-1,1]$ is a monotonic, increasing function satisfying $f(0)=0$. It is easy to note that computing the $\mathbf{F}_i(t)$ matrix does not increase the computational complexity of the TSE algorithm. To define the matrix $\mathbf{F}_i(t)$ one has only to compute the values $f(|\hat{d}_i(t) - \hat{d}_j(t)|)$, for any $j=1,2,\dots,r$, as in the case of a scalar form representation. Furthermore, as we shall see, the actual implementation does not necessarily utilize the vector conceptualization.

To preserve $\mathbf{P}(t)$ as a probability vector, we introduce a *weight* matrix $\mathbf{V}(t)$ as follows:

$$\begin{aligned} v_{ij}(t) &= 0, \text{ if } j \neq i, \\ v_{ij}(t) &= \frac{p_i(t)}{r-1}, \text{ if } \hat{d}_j(t) > \hat{d}_i(t), \\ v_{ij}(t) &= 1, \text{ if } \hat{d}_j(t) < \hat{d}_i(t), \\ v_{ii}(t) &= 1. \end{aligned} \quad (5.3)$$

Using the above-defined matrices, the updating equations for the TSE algorithm can be expressed as follows:

$$\mathbf{P}(t+1) = \mathbf{P}(t) + \lambda \cdot \mathbf{F}(t) \cdot \mathbf{V}(t) \cdot (\mathbf{e}(t) - \mathbf{P}(t)) \quad (5.4)$$

or, equivalently:

$$\mathbf{P}(t+1) = (\mathbf{I} - \lambda \cdot \mathbf{F}(t) \cdot \mathbf{V}(t)) \cdot \mathbf{P}(t) + \lambda \cdot \mathbf{F}(t) \cdot \mathbf{V}(t) \cdot \mathbf{e}(t) \quad (5.5)$$

where $\mathbf{I}$ is the identity matrix of dimensions $r \times r$.

The vectorial representation of the TSE algorithm⁶, Eq. (5.4), shows that, conceptually, the TSE algorithm moves the action probability vector $\mathbf{P}(t)$ towards the actions with higher estimates than the estimate of the current chosen action. This movement is done proportional to the distance between the current action probability vector and the potential solutions $\mathbf{e}(t)$, and it involves a learning parameter λ , a factor of the distance between the estimates, the matrix $\mathbf{F}(t)$, and a weight matrix $\mathbf{V}(t)$.

In this form, the updating equations of the TSE algorithm can easily be compared with the following updating equation of the Continuous Reward-Penalty Pursuit Algorithm.

$$\mathbf{P}(t+1) = \mathbf{P}(t) + \lambda \cdot (\mathbf{e}_m(t) - \mathbf{P}(t)) \quad (5.6)$$

It is easy to see that the vectorial representation of the TSE algorithm leads to various physical interpretations of the convergence process of the TSE algorithm. First of all, the TSE algorithm “extends” the Pursuit algorithms by pursuing more than one action, *i.e.* it pursues all the actions with higher estimates. Furthermore, the equation (5.4) shows that the TSE algorithm considers three more factors in its updating equations: a function of the distance between the estimates, a matrix $\mathbf{F}(t)$, and a weight matrix $\mathbf{V}(t)$.

⁶ Although we can represent the updating equations of the TSE algorithm in vector form, the implementation of the algorithm can be done in a scalar fashion, as will be explained presently.

This vectorial representation of the TSE algorithm sets the basis for permitting various generalizations. The following section presents one such generalization of the TSE algorithm.

5.2 Generalization of the TSE algorithm

This section presents a class of algorithms that generalize the TSE algorithm by permitting variations on the weight matrix $V(t)$, described above. The purpose of the *weight matrix* $V(t)$ is to ensure that the action probability vector $P(t)$ is preserved as a probability vector. When all the reward estimates are higher than the reward estimate of the chosen action α_i , the automaton increases all the probabilities $p_j(t+1)$, $j \neq i$. To ensure that the amount by which these probabilities increase does not surpass the value of $p_i(t)$, Thathachar and Sastry equally distributed the value of $p_i(t)$ to all $r-1$ action with higher estimates. Hence, the increment of each probability $p_j(t)$ is multiplied with $\frac{p_i(t)}{r-1}$.

In this section, we present a generalized version of the TSE algorithm, denoted by **GTSE**. In GTSE each increasing probability $p_j(t)$, $j=1, \dots, K(t)$ is scaled with a factor $p_i(t) \cdot \gamma_j(t)$, where $\gamma_j(t)$, $j=1, \dots, K(t)$ are defined such that

$$\gamma_1(t) + \gamma_2(t) + \dots + \gamma_{K(t)}(t) = 1, \quad 0 < \gamma_j(t) < 1, j=1, \dots, K(t) \quad (5.7)$$

and $K(t)$ represents the number of actions with higher reward estimates than the chosen action at time 't'.

Considering this, the weight matrix $V(t)$ is defined as:

$$\begin{aligned}
v_{ij}(t) &= 0, & \text{if } j \neq i, \\
v_{ij}(t) &= \gamma_j(t) \cdot p_i(t), & \text{if } \hat{d}_j(t) > \hat{d}_i(t), \\
v_{ij}(t) &= 1, & \text{if } \hat{d}_j(t) \leq \hat{d}_i(t).
\end{aligned} \tag{5.8}$$

The TSE algorithm becomes then a particular case of the GTSE algorithm, when $\gamma_j(t)$ takes the following value:

$$\gamma_j(t) = \frac{1}{r-1}, \text{ for all } j=1, \dots, K(t). \tag{5.9}$$

Note that in this case the sum of all the factors is less or equal to 1:

$$\gamma_1(t) + \gamma_2(t) + \dots + \gamma_{K(t)}(t) \leq 1$$

Moreover, the factor $\gamma_j(t)$ could be extended such that it represents the *a priori* information about the action probabilities [14], but, at this time, this problem remains open.

This single generalization leads us to a new learning strategy, described in detail below.

ALGORITHM GTSE

Parameters

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

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

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

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

$S_{ij}(t)$ an indicator function $S_{ij}(t) = \begin{cases} 1 & \text{if } \hat{d}_i(t) > \hat{d}_j(t) \\ 0 & \text{if } \hat{d}_i(t) \leq \hat{d}_j(t) \end{cases}$

f $f: [-1, 1] \rightarrow [-1, 1]$ a monotonic, increasing function satisfying $f(0)=0$

Method

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

Initialize $\hat{\mathbf{d}}(t)$ by picking each action a small number of times.

Repeat

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

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

$$\begin{aligned} p_j(t+1) &= p_j(t) - \lambda \cdot f(\hat{d}_i(t) - \hat{d}_j(t)) \cdot [S_{ij}(t)p_j(t) + S_{ji}(t) \cdot p_i(t) \cdot \gamma_j(t)(1 - p_j(t))] \quad j \neq i \\ p_i(t+1) &= p_i(t) + \lambda \cdot \sum_{j \neq i} f(\hat{d}_i(t) - \hat{d}_j(t)) \cdot [S_{ij}(t)p_j(t) + S_{ji}(t) \cdot p_i(t) \cdot \gamma_j(t)(1 - p_j(t))] \end{aligned} \quad (5.10)$$

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

$$\begin{aligned} W_i(t+1) &= W_i(t) + (1 - \beta(t)) \\ Z_i(t+1) &= Z_i(t) + 1 \\ \hat{d}_i(t+1) &= \frac{W_i(t+1)}{Z_i(t+1)} \\ \left. \begin{aligned} W_j(t+1) &= W_j(t) \\ Z_j(t+1) &= Z_j(t) \\ \hat{d}_j(t+1) &= \hat{d}_j(t) \end{aligned} \right\} \text{ , for all } j \neq i \end{aligned}$$

End Repeat

END ALGORITHM GTSE

As in the case of the TSE algorithm, this generalized version can be proven to be ϵ -optimal in any stationary environment. The proof for the ϵ -optimality of the GTSE algorithm follows the same idea as the TSE algorithm. First, it can be shown that using a sufficiently small value for the learning parameter λ , all actions are chosen enough number of times so that $\hat{\mathbf{d}}_m(t)$ will remain the maximum element of the estimate vector $\hat{\mathbf{d}}(t)$ after a finite time. Formally, this is be stated and proved below.

Theorem 5.1: For any given constants $\delta > 0$ and $M < \infty$, there exist $\lambda^* > 0$ and $t_0 < \infty$ such that under the GTSE algorithm, for all $\lambda \in (0, \lambda^*)$,

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

Proof:

Let us define the random variable Y_t^i as the number of times the i^{th} action was chosen up to time 't' in any specific realization. We must prove that $\Pr\{Y_t^i > M\} \geq 1 - \delta$, which is equivalent to

$$\Pr\{Y_t^i \leq M\} < \delta. \quad (5.11)$$

The events $\{Y_t^i = k\}$, $\{Y_t^i = j\}$ are mutually exclusive for $j \neq k$, and so (5.11) simplifies to:

$$\sum_{k=1}^M \Pr\{Y_t^i = k\} \leq \delta \quad (5.12)$$

We shall prove now that if the action α_i is chosen, then the action probability $p_i(t)$ satisfies the following inequality:

$$p_i(t) \geq (1 - \lambda) \cdot p_i(t-1) \quad (5.13)$$

From the updating equation (Eq.(5.10)), at any step 't' in the algorithm if the action α_i is chosen, the action probability $p_i(t)$ becomes:

$$p_i(t) = p_i(t-1) + \lambda \cdot \sum_{j \neq i} f(\hat{d}_i(t-1) - \hat{d}_j(t-1)) \cdot [S_{ij}(t-1)p_j(t-1) + S_{ji}(t-1) \cdot p_i(t-1) \cdot \gamma_j(t-1)(1 - p_j(t-1))]$$

The action α_i is chosen based on its action probability at time 't-1', and, therefore it can have any reward estimate value $\hat{d}_i(t-1)$. Specifically, the chosen action α_i could either

be the action with the maximum reward estimate, or the action with the minimum reward estimate, or one of the actions with an intermediate value of its reward estimate, as expressed by the following cases:

$$\text{Case 1: } \hat{d}_i(t-1) = \min_{j=1, \dots, r} \{\hat{d}_j(t-1)\}$$

$$\text{Case 2: } \hat{d}_i(t-1) = \max_{j=1, \dots, r} \{\hat{d}_j(t-1)\}$$

$$\text{Case 3: } \min_{j=1, \dots, r} \{\hat{d}_j(t-1)\} < \hat{d}_i(t-1) < \max_{j=1, \dots, r} \{\hat{d}_j(t-1)\}$$

Therefore, the action probability $p_i(t)$ can have a range of values, depending on which action is chosen.

We proceed to prove the inequality (5.13). If any action $\alpha(t)=\alpha_i$, $i=1, \dots, r$ is chosen, it is sufficient to prove that the minimum possible value of the action probability of the chosen action, $p_i(t)$, satisfies this inequality. Based on the updating equations (5.10) it is easy to see that $p_i(t)$ achieves the minimum value when the chosen action represents the action with the minimum reward estimate, as described in Case 2. In the other two cases, Case 1 and Case 3, there are some actions with smaller reward estimates than the chosen action, and therefore, the action probabilities of these actions decrease. The amount with which these probabilities decrease is added to the action probability $p_i(t)$, hence in these cases $p_i(t)$ does not achieve the minimum value. In Case 2, the action probabilities increase for all the actions α_j , $j \neq i$ and the action probability $p_i(t)$ becomes the minimum possible value of $p_i(t)$. This value is calculated to be:

$$\begin{aligned}
p_i^{\min}(t) &= \min\{p_i(t)\} = \\
&= p_i(t-1) - \lambda \cdot \sum_{j \neq i} \left[f(\hat{d}_j(t-1) - \hat{d}_i(t-1)) \right] \cdot [p_i(t-1) \cdot \gamma_j(t-1)(1 - p_j(t-1))]
\end{aligned}$$

Because $f(x) \leq 1$, and $p_j(t) \leq 1$, this implies:

$$\begin{aligned}
p_i^{\min}(t) &\geq p_i(t-1) - \lambda \cdot \sum_{j \neq i} p_i(t-1) \cdot \gamma_j(t-1) \cdot (1 - p_j(t-1)) \\
&\geq p_i(t-1) - \lambda \cdot \sum_{j \neq i} p_i(t-1) \cdot \gamma_j(t-1) \\
&= p_i(t-1) - \lambda \cdot p_i(t-1) \cdot \sum_{j \neq i} \gamma_j(t-1)
\end{aligned} \tag{5.14}$$

From the definition of the $\gamma_j(t)$ factors (Eq. (5.7)) we know that the sum of these factors is unity, which implies that

$$p_i^{\min}(t) \geq p_i(t-1) - \lambda \cdot p_i(t-1) = (1 - \lambda) \cdot p_i(t-1) . \tag{5.15}$$

This proves that

$$p_i(t) \geq p_i^{\min}(t) \geq (1 - \lambda) \cdot p_i(t-1) \tag{5.16}$$

which implies that during any of the first 't' iterations of the algorithm

$$\Pr\{\alpha_i \text{ is chosen}\} \geq p_i(0) \cdot (1 - \lambda)^t, \text{ for any } i=1, \dots, r \tag{5.17}$$

Also, at any iteration of the algorithm, $\Pr\{\alpha_i \text{ is chosen}\} \leq 1$.

The reminder of the proof is identical to the proof for the TSE algorithm and omitted. It can be found in [28].

◆◆◆

The next step in proving the ε -optimality of the GTSE algorithm consists of demonstrating that if there is an action α_m for which the reward estimate remains maximal after a finite number of iterations, then the m^{th} component of the action

probability vector converges in probability to 1. This result is formally proven in the next theorem.

Theorem 5.2: Suppose that there exists an index m and a time instant $t_0 < \infty$ such that

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

then $p_m(t) \rightarrow 1$ with probability 1 as $t \rightarrow \infty$.

Proof:

To prove this result, we shall demonstrate that the sequence of random variables $\{p_m(t)\}_{t \geq 0}$ is a submartingale. The convergence in probability will result then from the submartingale convergence theorem [11].

Based on the assumptions of the theorem, $p_m(t+1)$ can take the following values:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \lambda \cdot \sum_{j \neq m} f(\hat{d}_m(t) - \hat{d}_j(t)) \cdot p_j(t) \cdot \gamma_j(t) \cdot (1 - p_m(t)) & \text{if } \alpha_j \text{ is chosen} \\ p_m(t+1) &= p_m(t) + \lambda \cdot \sum_{j \neq m} f(\hat{d}_m(t) - \hat{d}_j(t)) \cdot p_j(t) & \text{if } \alpha_m \text{ is chosen} \end{aligned} \quad (5.18)$$

From these equations we calculate:

$$\begin{aligned} \Delta p_m(t+1) &= E[p_m(t+1) - p_m(t) | Q(t)] \\ &= \lambda \sum_{j \neq m} [f(\hat{d}_m(t) - \hat{d}_j(t)) \cdot p_j(t) \cdot \gamma_j(t) (1 - p_m(t))] \cdot p_j(t) \\ &\quad + p_m \cdot \lambda \cdot \sum_{j \neq m} [f(\hat{d}_m(t) - \hat{d}_j(t)) \cdot p_j(t)] \cdot p_m(t) \\ &= \lambda \cdot \sum_{j \neq m} f(\hat{d}_m(t) - \hat{d}_j(t)) [p_j^2(t) \cdot \gamma_j(t) \cdot (1 - p_m(t)) + p_j(t) \cdot p_m(t)] \geq 0 \end{aligned} \quad (5.19)$$

Since $\Delta p_m(t+1)$ for all $t \geq t_0$, implies that that $p_m(t)$ is a submartingale. By the submartingale convergence theorem [11], $\{p_m(t)\}_{t \geq 0}$ converges as $t \rightarrow \infty$,

$E[p_m(t+1) - p_m(t) \mid Q(t)] \rightarrow 0$ with probability 1.

Hence, $p_m(t) \rightarrow 1$ with probability 1, and the theorem is proven. $\diamond \diamond \diamond$

Finally, the Theorem 5.3 states the ε -optimality convergence for the GTSE algorithm, and can be easily derived from the two previous results.

Theorem 5.3: For the GTSE algorithm, in every stationary random environment, there exists a $\lambda^* > 0$ and $t_0 > 0$, such that for all $\lambda \in (0, \lambda^*)$ and for any $\delta \in (0, 1)$ and any $\varepsilon \in (0, 1)$,

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

for all $t > t_0$. $\diamond \diamond \diamond$

This proves that this generalized TSE algorithm is ε -optimal in every stationary environment.

The next section presents simulation results for the GTSE algorithm

5.3 Simulation Results

This section presents simulation results performed to compare the performance of the TSE algorithm with a specific version of the Generalized TSE algorithm. In simulating the GTSE algorithm, the $\gamma_j(t)$ has been assigned to be $1/K(t)$, where $K(t)$ is the number of actions with higher estimates than the chosen action $\alpha_i(t)$ at time 't', or 1 if the chosen action is the one with the highest reward estimate.

The simulation environments that have been considered are the same benchmark environments as the ones for which the simulations were described in Sections 3.4 and

4.4. Specifically, the first set of simulations was performed in order to determine the optimal value for the learning parameter for each of these algorithms. Then, using these learning parameters, each algorithm was required to converge in 750 experiments and the average number of iterations was computed. For each algorithm, 100 test were performed and the number of iterations presented in this section is the average over these experiments.

For the two-action environments, the weight matrix $V(t)$ for the TSE algorithm is identical with the weight matrix for the GTSE algorithm. Therefore, simulations were performed only for the benchmark ten-action environments E_A and E_B . Table 5.1 presents the results obtained in these environments.

Table 5.1: Performance of the GTSE and TSE algorithms in ten-action benchmark environments for which exact convergence was required in 750 experiments ($NE=750$)

Environ- ment	GTSE		TSE		CP _{RP}		CP _{RI}	
	λ	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.	λ	No. of Iterat.
E_A	0.068	860.39	0.064	913.99	0.003	2427.3	0.0097	1230.3
E_B	0.038	2485.69	0.035	2745.64	0.00126	5685	0.002	4603

Note: The reward probabilities for the actions are:

E_A : 0.7 0.5 0.3 0.2 0.4 0.5 0.4 0.3 0.5 0.2

E_B : 0.1 0.45 0.84 0.76 0.2 0.4 0.6 0.7 0.5 0.3

The simulations results show that considering $\gamma_j(t)=1/K(t)$, where $K(t)$ is the number of actions with higher estimates than the chosen action at time 't', the generalized TSE algorithm exhibits an increase in performance of up to 9% relative to the performance of

the TSE algorithm. For example, in the benchmark environment E_B , the TSE algorithm required for convergence on average 2745.64 iterations, whereas the GTSE algorithm required on average 2485.69 iterations, being 9% faster than the TSE algorithm. Also, if compared against the class of Pursuit algorithms, the GTSE algorithm proves to be faster than the CP_{RP} and CP_{RI} algorithm, and faster than the generalized Pursuit Algorithm GPA. For example, in the E_A environment, the GTSE algorithm requires on average 860.39 iterations, whereas the CP_{RI} requires on average 1230.3 iterations, CP_{RP} requires on average 2427.3 iterations for convergence, and GPA requires 948.03.

The following chart shows the improvement of the performance of the existing continuous estimator algorithms relative to the performance of the CP_{RP} algorithm.

Figure 5.1: Performance of the continuous Estimator Algorithms relative to the CP_{RP} algorithm in ten-action environments for or which exact convergence was required in 750 experiments

It can easily be seen that based on these experimental results, the GTSE algorithm proves to be the fastest continuous estimator algorithm in benchmark ten-action environments.

Although the GTSE is a continuous algorithm, simulations were executed to compare its performance against the discretized TSE algorithm (DTSE). To obtain the simulation results for the DTSE algorithm, we first had to determine the set of learning parameters that led to the best performance of the algorithm. Unlike the other estimator algorithms, the DTSE algorithm has two learning parameters: N , the resolution parameter, and θ . Therefore, we had to determine both N and θ for which the DTSE algorithm exhibits the best performance when required to converge in NE experiments. Rather than searching the entire (N, θ) parameter space, we have considered just few values of θ ($\theta=2, \theta=10, \theta=100$), and determined the optimal value of N for these values of θ . Although the results do not span the entire parameter space, it is clear that the results are both typical and conclusive. The best performance of the DTSE algorithm was obtained with $\theta=2$, and the simulation results are presented in Table 5.2.

Table 5.2: Performance of the GTSE and DTSE algorithms in benchmark ten-action environments for which exact convergence was required in 750 experiments (NE=750)

Environment	GTSE		DTSE	
	λ	No. of Iterat.	N	No. of Iterat.
E _A	0.068	860.39	99	518.18
E _B	0.038	2485.69	248	1126.86

Note: DTSE results were obtained with $\theta=2$.

From these results, one can notice that the DTSE algorithm is up to 50% faster than the GTSE algorithm. For example, in the E_B environment, the GTSE algorithm required on average 2485.69 iterations to converge whereas the DTSE algorithm required on average 1126.86 iterations.

The problem of extending the GTSE algorithm into the discretized world remains open.

5.4 Conclusions

Thathachar and Sastry introduced the TSE algorithm [29] and presented it in a scalar form. In this chapter, we introduced a vectorial representation of the TSE algorithm, which led to a better conceptual perception of the TSE algorithm. This new vectorial form of the TSE algorithm also provided the background for generalizing the TSE algorithm by extending the weight matrix $V(t)$.

A class of generalized TSE algorithms, GTSE, has been presented in this chapter, along with the proof of their ϵ -optimality. Simulation results have been presented for a specific GTSE algorithm which considered the factors $\gamma_i(t)=1/K(t)$, where $K(t)$ is the number of actions with higher estimates than the chosen action at time 't'. Although the GTSE algorithm can be represented in vector form, its implementation has been done in scalar form, as presented in the detailed description of the algorithm.

Based on the experimental results, we submit that the new GTSE algorithm is superior to the TSE algorithm. In the ten-action benchmark environments, it exhibits an increased performance of up to 9% when compared against the well-known TSE algorithm. Furthermore, if compared against all the existing continuous estimator algorithms, the GTSE algorithm proves to be the fastest one.

The generalization of the TSE algorithm has been presented only in a continuous probability space. Its version in a discretized probability space remains an open problem.

We conclude this section by mentioning that the novel results of this chapter are currently being compiled into a potential publication.

Chapter 6: CONCLUSIONS

6.1 Summary

In this thesis, we have presented a study of the reported and of some novel Estimator Algorithms in Learning Automata. The thesis started by presenting an overview of the field of learning automata and of some of the most important existing algorithms. Tsetlin [31] first introduced the concept of a learning automaton by designing a Fixed Structure Stochastic Automata (FSSA) traditionally known as the Tsetlin Automaton. Chapter 2 presented this automaton along with other FSSA automata. We then introduced the family of Variable Structure Stochastic Automata (VSSA), and we explained various such learning automata (e.g. L_{RI} , L_{RP} , L_{IP}). The discretization concept was then highlighted, and the discretized versions of the continuous VSSA algorithms were presented. Finally, Chapter 2 also explained the Estimator Algorithms, by describing some of the existing continuous and discrete Estimator Algorithms such as CP_{RP} , TSE, DP_{RI} and DTSE schemes.

The thesis continued by concentrating on the study of the Pursuit Estimator Algorithms. In Chapter 3, we presented new versions of the Pursuit algorithms that

resulted from the combination of the Reward-Penalty and Reward-Inaction learning paradigms with the continuous and discrete models of computation. Just as in the case of Pursuit automata, these new algorithms were shown to be ϵ -optimal in all random environments. Also, in this chapter we presented a performance comparison of the newly introduced algorithms, the CP_{RI} and DP_{RP} , with the already existing versions of Pursuit algorithms, the CP_{RP} and DP_{RI} . Using these simulation results, we ranked these Pursuit algorithms based on their performance as follows:

Best Algorithm: Discretized Pursuit Reward-Inaction (DP_{RI})

2nd-best Algorithm: Discretized Pursuit Reward-Penalty (DP_{RP})

3rd-best Algorithm: Continuous Pursuit Reward-Inaction (CP_{RI})

4th-best Algorithm: Continuous Pursuit Reward-Penalty (CP_{RP})

In Chapter 4, we continued the study of the Pursuit estimator algorithms by considering two generalized versions of the CP_{RP} Pursuit algorithm. These new algorithms generalize the Pursuit concept by attempting to linearly ‘pursue’ not only the action with the maximal estimate, but also all the actions with higher reward estimates. The new automaton, referred to as the GPA automaton, is an example of a continuous generalized Pursuit algorithm. Using these same concepts, we then presented a discretized version, DGPA, of a generalized Pursuit algorithm. Regarding their convergence properties, we proved that these two algorithms are ϵ -optimal. As part of the evaluation of these new algorithms, simulations were performed to determine their performance. Based on the experimental results obtained, we concluded that the **GPA algorithm is the fastest-converging continuous Pursuit Algorithm**, and that the

DGPA is the fastest discrete Pursuit algorithm, and, more general, the **fastest Pursuit algorithm**. From the performance perspective, we ranked the Pursuit algorithms as follows:

Best Algorithm: Discretized Generalized Pursuit Algorithm (DGPA)

2nd-best Algorithm: Discretized Pursuit Reward-Inaction (DP_{RI})

3rd-best Algorithm: Generalized Pursuit Algorithm (GPA)

4rd-best Algorithm: Discretized Pursuit Reward-Penalty (DP_{RP})

5rd-best Algorithm: Continuous Pursuit Reward-Inaction (CP_{RI})

6th-best Algorithm: Continuous Pursuit Reward-Penalty (CP_{RP})

In Chapter 5, we concentrated on the study of the TSE estimator algorithm, introduced by Thathachar and Sastry. We first presented a novel vectorial representation of the updating equations of this algorithm. This vectorial form (Eq. (5.4)) outlines the concepts used by the TSE algorithm in the learning process, and it allows for an easy comparison of the TSE algorithm with the CP_{RP} algorithm. Furthermore, this vectorial representation set the basis for permitting various generalizations of the TSE algorithm. In this chapter we formulated a generalized TSE algorithm, referred to as GTSE, based on variations of the weight matrix $V(t)$ which appears in the vectorial representation. The GTSE algorithm has been presented only in the continuous probability space, its discretization remaining an open problem. Regarding its convergence, we proved that this algorithm is *ϵ -optimal* in every stationary environment. To complete the characterization of the newly introduced GTSE algorithm, we presented simulation

results, which show that the GTSE algorithm is the **fastest reported continuous estimator algorithm**.

In conclusion, in this thesis we have introduced five new estimator algorithms: CP_{RI} , DP_{RP} , GPA, DGPA and GTSE. The following graph summarizes the performance improvement of all these estimator algorithms relative to the performance of the CP_{RP} algorithm.

Figure 6.1: Performance of some Estimator Algorithms relative to the CP_{RP} algorithm in ten-action environments for which exact convergence was required in 750 experiments

Based on the experimental results presented in this thesis, we conclude that among all the algorithms that we have analyzed, the newly introduced GPA algorithm is the fastest continuous Pursuit algorithm, the DGPA is the fastest converging discretized Pursuit estimator algorithm, and, indeed, the fastest Pursuit estimator algorithm. In the

class of the continuous estimator algorithms, the GTSE proved to be the fastest continuous estimator algorithm.

6.2 Future work

In this thesis, the class of Estimator Algorithms has been extensively studied and various new schemes have been introduced. In spite of this, this study proved that these algorithms can be expanded. Furthermore, opens some new research directions.

One interesting direction that should be followed is to try to incorporate *a priori* information in the weight matrix $V(t)$ of the generalized TSE (GTSE) algorithm. Additionally, since a generalization of the TSE algorithm has been presented only in the continuous probability space, future research could concentrate on generalizing the TSE algorithm also in the discrete space.

REFERENCES

- [1] M. Agache, B.J. Oommen, "*Continuous and Discretized Generalized Pursuit Learning Schemes*", 4th World Multiconference on Systemics, Cybernetics and Informatics, July 23-26, Orlando, Florida.
- [2] K. S. Fu, "*Learning Control Systems – Review and outlook*", IEEE Trans. on Automatic Control, Vol. 15, pp. 210-221, April, 1970.
- [3] K. S. Fu, "*Learning Control Systems and Intelligent Control Systems: An Intersection of Artificial Intelligence and Automatic Control*", IEEE Trans. on Automatic Control, Vol. 16, pp. 70-72, 1971.
- [4] K. S. Fu, "Pattern Recognition and Machine Learning", Plenum Press, 1971.
- [5] K. Fukunaga, "Introduction to Statistical Pattern Recognition", Academic Press, 1972.
- [6] D.L. Isaacson, R.W. Madsen, "Markov Chains: Theory and Applications", New York: John Wiley&Sons, 1976.
- [7] S. Lakshmivarahan, "Learning Algorithms Theory and Applications", New York: Springer-Verlag, 1981.
- [8] S. Lakshmivarahan, M.A.L. Thathachar, "*Absolutely Expedient Learning Algorithms for Stochastic Automata*", IEEE Trans. Syst., Man and Cybern., SMC-3, 1973, pp.281-286.
- [9] J.K. Lanctôt, "*Discrete estimator algorithms: A mathematical model of computer learning*", M.Sc. Thesis, Dept. Math. Statistics, Carleton Univ., Ottawa, Canada, 1989.
- [10] J.K. Lanctôt, B. J. Oommen, "*Discretized Estimator Learning Automata*", IEEE Trans. on Syst. Man and Cybernetics, Vol. 22, No. 6, pp. 1473-1483, November/December 1992.
- [11] K.S. Narendra, M.A.L. Thathachar, "Learning Automata an Introduction", Prentice-Hall, 1989.
- [12] K.S. Narendra, M.A.L. Thathachar, "*Learning Automata – A survey*", IEEE Trans. on Syst. Man and Cybernetics, Vol. SMC-4, 1974, pp.323-334.

- [13] K.S. Narendra, E. A. Wright, L. G. Mason, "*Application of Learning Automata to Telephone Traffic Routing and Control*", IEEE Trans. on Sys., Man., and Cybern., Vol. SMC-7, No. 11, November 1977.
- [14] B.J. Oommen, "*Absorbing and Ergodic Discretized Two-Action Learning Automata*", IEEE Trans. Syst. Man Cybern., vol SMC-16, no. 4, pp.282-2996, Mar./Apr. 1986.
- [15] B.J. Oommen, "*A Learning Automaton Solution to the Stochastic Minimum-Spanning Circle Problem*", IEEE Trans. Syst. Man Cybern., vol SMC-16, no. 2, pp.282-2996, July/August 1986.
- [16] B.J. Oommen and M. Agache, "*A Comparison of Continuous and Discretized Pursuit Learning Schemes*", IEEE. International Conference on Syst. Man. Cybern., October 12-19, 1999, Tokyo, Japan.
- [17] B.J. Oommen and J. P. Christensen, "*Epsilon-optimal discretized reward-penalty learning automata*", IEEE. Trans. Syst. Man. Cybern., vol. SMC-18, pp. 451-458, May/June 1988.
- [18] B.J. Oommen and J.K. Lanctôt, "*Discretized Pursuit Learning Automata*", IEEE Trans. Syst. Man. Cybern., vol. 20, No.4, pp.931-938, July/August 1990.
- [19] B.J. Oommen and E.R. Hansen, "*The asymptotic optimality of discretized linear reward-inaction learning automata*", IEEE. Trans. Syst. Man. Cybern., pp. 542-545, May/June 1984.
- [20] B.J. Oommen and G. Raghunath, "*Automata Learning and Intelligent Tertiary Searching for Stochastic Point Location*", IEEE. Trans. Syst. Man. Cybern. – Part B, Cybernetics, Vol. 28, No.6, pp. 947-954, December 1998.
- [21] G.I. Papadimitriou, "*A New Approach to the Design of Reinforcement Schemes for Learning Automata: Stochastic Estimator Learning Algorithms*", IEEE Trans. on Knowledge and Data Eng., Vol. 6, No. 4, pp. 649-654, August 1994.
- [22] A. Paz, "*Introduction to Probabilistic Automata*", New York Academic Press, 1971.
- [23] J. Slagle, "*Artificial Intelligence and Heuristic Programming*", McGraw Hill, 1975.
- [24] R. S. Sutton, A. G. Barto, "*Reinforcement Learning – An Introduction*", The MIT Press, Cambridge, Massachusetts, 1998.
- [25] R. Solomonoff, "*Some Recent Work in Artificial Intelligence*", Proc. of IEEE, Vol. 54, Dec. 1966.
- [26] M.A.L. Thathachar and B.J. Oommen, "*Discretized reward-inaction learning automata*", J. Cybern. Information Sci., pp.24-29, Spring 1979.

- [27] M.A.L. Thathachar and P.S. Sastry, "*Pursuit Algorithm for Learning Automata*" Transcript provided by the author.
- [28] M.A.L. Thathachar and P.S. Sastry, "*A class of rapidly converging algorithms for learning automata*", presented at IEEE Int. Conf. on Cybernetics and Society, Bombay, India, Jan. 1984.
- [29] M.A.L. Thathachar and P.S. Sastry, "*A New Approach to the Design of Reinforcement Schemes for Learning Automata*" IEEE Trans. Syst., Man, Cybern., vol SMC-15, No. 1, pp. 168-175, January/February 1985.
- [30] M.A.L. Thathachar and P.S. Sastry, "*Learning Optimal Discriminant Functions Through a Cooperative Game of Automata*", IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-17, No.1, January/February 1987.
- [31] M.L. Tsetlin, "*On the behavior of finite automata in random media*", Automat. Telemek. (USSR), vol. 22, pp. 1345-1354, Oct. 1961.
- [32] M.L. Tsetlin, "*Automaton Theory and the Modeling of Biological Systems*", New York: Academic, 1973.
- [33] R. Viswanathan and K.S. Narendra, "*Stochastic Automata Models with Application to Learning Systems*", IEEE Trans. Syst., Man. and Cyber., SMC-3, 1973, pp. 107-11.
- [34] V.I. Varshavskii and I.P. Vorontsova, "*On the behavior of stochastic automata with variable structure*", Automat. Telemek. (USSR), vol. 24, pp. 327-333, 1963.