1

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

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

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

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

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

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed

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

AVIS

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

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

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

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

Object Partitioning By Using Learning Automata

by

DANIEL CHIU YU MA, B. Math:

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

School of Computer Science
Carleton University
Ottawa, Ontario

1st April, 1986.

© Copyright

1986, Daniel Chiu Yu Ma

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-29879-0

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

OBJECT PARTITIONING BY USING LEARNING AUTOMATA

submitted by Daniel Chiu Yu'Ma, B. Math.

in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chairman, School of Computer Science

Carleton University

Date

Abstract

The problem of partitioning a set of W objects $A = \{A_1, \dots, A_W\}$ into R classes $\{P_1, \dots, P_R\}$ is studied. The intention is to partition the objects such that those that are accessed more frequently together are located in the same class. The joint access probabilities between objects are unknown. In this thesis, we consider two cases of the problem, namely the case when the classes are of equal size (the Equal Partitioning Problem) and the case when they are of unequal size. In the former case, we propose three deterministic learning automata solutions. Although the first two solutions are ϵ -optimal, they seem to be feasible only when W is small. The third one, which uses a new learning automaton, demonstrates an excellent partitioning capability and is feasible even for large values of W . Experimentally, this solution converges much faster than the best known algorithm in the literature [27]. For the case of non-equipartitioning, we propose a new variable structure stochastic automaton solution. This solution uses an updating scheme derived from the well-known Linear Reward-Penalty scheme and is shown to be expedient. Experimental results seem to indicate its ϵ -optimality.

Acknowledgements

I wish to express my gratitude to God for giving me a good thesis supervisor and for helping me finish this thesis.

To my supervisor, Professor B. John Oommen, I would render my thankfulness for his concern for me, for his encouragement when I needed it, and for the time he spent on the discussions related to this thesis.

I am also grateful to my fellow students, especially to Rod Nickel for his comments and time spent on reading my thesis, to Terry Copeck and George Wang for their stimulating discussions, and to Hin Wai Chang for his assistance.

Finally, I would like to thank my father, my sister, and the Carleton University for the financial support of my studies.

Table of Contents

Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	ix
List of Figures	xi
1. Introduction	1
1.1 Object Partitioning Problem	2
1.2 Applications of the Object Partitioning Problem	4
1.3 A Related Problem in the Literature	6
1.4 Previous Solutions and Our Proposed Strategy	8
1.5 Thesis Outline	9
2. Previous Work	12
2.1 A "Hill-Climbing" Technique	13
2.1.1 The Method	13
2.1.2 Intuition of the Heuristic	15
2.1.3 Experimental Results	15
2.2 The Basic Adaptive Method	16
2.2.1 Abstract Objects	17
2.2.2 Convention	17
2.2.3 The BAM Algorithm	18
2.2.4 An Illustrative Example	20
2.2.5 The Between-Class and Within-Class Distances	23
2.2.6 Cluster Identification	24
2.2.7 Experiments Conducted.....	25
2.2.7.1 Initialization	25
2.2.7.2 Parameters	26
2.2.7.3 Assumptions	26
2.2.7.4 Results	27
2.2.8 Complexity of the BAM	28
2.3 Concluding Remarks	30
3. Overview of Learning Automata	31
3.1 The Concept of Learning	31
3.2 Learning Automata Models	32
3.3 Random Environments	34

3.3	Random Environments	34
3.4	The Learning Objective	34
3.5	Learning Criteria	35
3.6	Classification of Learning Automata	35
3.6.1	Fixed Structure Stochastic Automata (FSSA)	36
3.6.1.1	The Learning Process of FSSA	36
3.6.1.2	An FSSA Example	37
3.6.2	Variable Structure Stochastic Automata (VSSA)	39
3.6.2.1	The Learning Process of VSSA	42
3.6.2.2	A VSSA Example	43
3.7	Concluding Remarks	44
4.	The Equal Partitioning Problem	45
4.1	Equal Partitioning Problem	45
4.2	The Tsetlin and the Krinsky Automata	46
4.3	The Tsetlin and the Krinsky Automata Solutions to the EPP	49
4.3.1	Approach	49
4.3.2	Structures of the Tsetlin and the Krinsky Learning Automata	50
4.3.3	The Environment	51
4.3.4	Method	51
4.3.5	A Tsetlin or Krinsky Automaton Example	55
4.3.6	Implementation of the Algorithms	57
4.3.7	Experiments	57
4.3.7.1	Initialization	58
4.3.7.2	Parameters and Assumptions	58
4.3.7.3	Convergence	59
4.3.7.4	Results	59
4.3.8	Comments	61
4.3.9	Complexities of the Tsetlin and the Krinsky Automata	62
4.4	The Object Migrating Automaton (OMA)	63
4.4.1	Motivation	63
4.4.2	Approach	63
4.4.3	Environment	64
4.4.4	Structure of the OMA	64
4.4.5	Method	65
4.4.6	An OMA Example	70
4.4.7	Analytical Results Concerning the OMA	73
4.4.8	Experiments	76
4.4.8.1	Initialization	77
4.4.8.2	Parameters and Assumptions	77
4.4.8.3	Results	77
4.4.9	Complexity of the OMA	81
4.5	Concluding Remarks	82

5. FSSA Approaches To The OPP	83
5.1 Generalization of the Tsetlin and the Krinsky Automata Solutions	83
5.1.1 Method	83
5.1.2 Problems with the Tsetlin and the Krinsky Approach	84
5.2 Generalization of the OMA Approach	84
5.2.1 Method	85
5.2.2 Problem	85
5.3 Two Important Properties	87
5.4 The Stochastic Move Automaton (SMA)	89
5.4.1 Approach	89
5.4.2 Structure of the SMA	90
5.4.3 Environment	91
5.4.4 Method	91
5.4.5 An SMA Example	94
5.4.6 Experiments	97
5.4.6.1 Initialization	97
5.4.6.2 Parameters and Assumptions	97
5.4.6.3 Convergence	98
5.4.6.4 Results	98
5.4.7 Complexity of the SMA	108
5.5 Comments	108
5.6 Concluding Remarks	109
6. A VSSA Approach To The OPP	110
6.1 Motivation	110
6.2 The Linear Reward-Penalty Scheme	112
6.3 The Modified Linear Reward-Penalty Scheme	113
6.3.1 Motivating Idea	113
6.3.2 Structure of the ML _{RP} Automaton	114
6.3.3 Environment	115
6.3.4 Implementation Detail	115
6.3.5 An ML _{RP} Example	118
6.3.6 Choice of the Parameters	121
6.3.6.1 Experimental Results for the Parameters	122
6.3.6.1.1 Experiments Performed in an Informative Environment	122
6.3.6.1.2 Experiments Performed in Less Information Environments	127
6.3.6.2 Comments	128
6.3.7 Experiments Comparison with the BAM	132
6.3.7.1 Initialization	132

6.3.7.2 Parameters and Assumptions	132
6.3.7.3 Convergence	133
6.3.7.4 Results	133
6.4 Comments	142
6.5 Complexity of the ML_{PP} Automaton	143
6.6 Concluding Remarks	144
7. Conclusions	145
7.1 Summary of the Results	145
7.2 Future Research	147
References	148

List of Tables

Table	Description	Page
1.1	Exponential Growth in the Number of Partitions	4
2.1	Rate of Convergence and Accuracy of the BAM	29
4.1	Comparison of the BAM and the Tsétlin and the Krinsky Automata Solutions for the EPP	60
4.2	Comparison of the BAM and the OMA Solutions for the EPP	79
4.3	Comparison of the BAM and the OMA Solutions for Various Joint Access Probabilities	81
5.1(a)	Comparison of the Rate of Convergence of the BAM and the SMA Solutions for the OPP, $\delta = 0.25$	100
5.1(b)	Comparison of the Rate of Convergence of the BAM and the SMA Solutions for the OPP, $\delta = 0.5$	101
5.2(a)	Comparison of the Accuracy of the BAM and the SMA Solutions for the OPP, $\delta = 0.25$	103
5.2(b)	Comparison of the Accuracy of the BAM and the SMA Solutions for the OPP, $\delta = 0.5$	104
5.3	Comparison of the Accuracy of the BAM and the SMA for Various Joint Access Probabilities	106
6.1	Comparison of the Rate of Convergence of the ML_{RP} Automaton for Various λ_1 and λ_2	124
6.2	Comparison of the Accuracy of the ML_{RP} Automaton for Various of λ_1 and λ_2	124

6.3	Comparison of the Rate of Convergence of the ML_{RP} Automaton for Different λ_1 , for Various Joint Access Probabilities	129
6.4	Comparison of the Accuracy of the ML_{RP} Automaton for Different λ_1 , for Various Joint Access Probabilities	129
6.5	Comparison of the Rate of Convergence of the BAM and the ML_{RP} Automaton Solutions	135
6.6	Comparison of the Accuracy of the BAM and the ML_{RP} Automaton Solutions	137
6.7	Comparison of the Rate of Convergence and the Accuracy of the BAM and the ML_{RP} Automaton as a Function of the Joint Access Probabilities	139

List of Figures

Figure	Description	Page
2.1	An Example of Using Only the First Step of the BAM	21
2.2	An Example of the BAM	22
3.1	The Automaton-Environment Interaction	33
3.2	An FSSA Example: The Tsetlin Automaton	38
4.1	$L_{2N,2}$, The Two-Action $2N$ -State Tsetlin Automaton	47
4.2	$Z_{2N,2}$, The Two-Action $2N$ -State Krinsky Automaton	49
4.3	The Object Migrating Automaton	67
4.4	Plot of the Rate of Convergence of the BAM and the OMA as a Function of the Size of the Object Set	80
5.1	Plot of the Rate of Convergence of the BAM and the SMA as a Function of the Size of the Object Set	102
5.2	Plot of the Accuracy of the BAM and the SMA as a Function of the Size of the Object Set	105
5.3	Plot of the Accuracy of the BAM and the SMA as a Function of the Joint Access Probability	107
6.1	Plot of the Rate of Convergence of the ML_{RP} Automaton as a Function of λ_1 , for $\lambda_2 = 0.975, 0.95, 0.9, 0.85$	125
6.2	Plot of the accuracy of the ML_{RP} Automaton as a Function of λ_1 , for $\lambda_2 = 0.975, 0.95, 0.9, 0.85$	126
6.3	Plot of the Rate of Convergence of the ML_{RP} Automaton as a Function of Joint Access Probability, for Different λ_1	130

6.4	Plot of the Accuracy of the ML_{RP} Automaton as a Function of Joint Access Probability, for Different λ_1	131
6.5	Plot of the Rate of Convergence of the BAM and the ML_{RP} Automaton as a Function of the Size of the Object Set	136
6.6	Plot of the Accuracy of the BAM and the ML_{RP} Automaton as a Function of the Size of the Object Set	138
6.7	Plot of the Rate of Convergence of the BAM and the ML_{RP} Automaton as a Function of Joint Access Probability	140
6.8	Plot of the Accuracy of the BAM and the ML_{RP} Automaton as a Function of Joint Access Probability	141

Chapter 1

Introduction

As the number of users of computers and the amount of data in database systems increase, it is necessary to organize data properly so that user's queries (transactions) can be answered efficiently. One possible way to achieve this goal is to reduce the amount of data needed to be accessed in processing individual queries. Attribute partitioning is a well-known database design technique whose aim is to improve the performance of database systems by reducing the amount of data which has to be accessed or processed.

Let us suppose the database designer is given a logical file consisting of a set of logical records and that each record consists of n attributes (fields). Obviously, if all the information is stored in one physical file, then a set of records containing all the attributes will be required for the processing of a query, even though the query may only request for a subset of these n attributes. This is undesirable when n is large or when the size of each record is large (i.e., a large amount of data is stored in each or some of the attributes). The concept of attribute partitioning, as described in [4], involves breaking the logical records of a logical file into several physical records stored in different physical subfiles so that each record contains only a subset of the attributes of the original record. If the subrecords are designed in such a way that the attributes needed to process typical queries are all located in the same subrecords, then, typically, only one physical file will have to be accessed. As a consequence of such a partitioning, the number of files and the amount of data which has to be accessed in processing queries will be reduced, and in

general, the database system performance will be improved.

Observe that although the physical file is partitioned into several subfiles, this partitioning is an issue tackled by the database management system and is completely **invisible** to the users. The users will continue to express their queries against the logical file, and the database system must be able to process such queries against the partitioned subfiles.

Generally speaking, the attribute partitioning problem [4] involves finding the partition of the attributes of a logical file based on the description of the logical file and a set of queries against it. The intention is to group them in such a way that the performance of the system is maximized in terms of query processing. More precisely, the concept of partitioning attributes involves the clustering of "similar" attributes (i.e., attributes that are usually accessed together). Simultaneously, it involves the divorcing of "dissimilar" attributes from each other, where, by "dissimilar" we imply attributes that are seldom accessed together.

The attribute partitioning problem and various other problems in a variety of disciplines have a marked similarity. Since we intend to tackle all these problems, it is appropriate to rephrase this problem in a more general framework.

1.1 Object Partitioning Problem

Let $\mathcal{A} = \{A_1, \dots, A_W\}$ be a set of W objects. We intend to partition $\mathcal{A}$ into R classes $\{P_1, \dots, P_R\}$, where each class may have a different number of objects. Further, we intend to partition these objects in such a way that the objects that are accessed (used) more frequently together lie in the same class. For example, suppose we are given a set of five objects $\{A_1, A_2, A_3, A_4, A_5\}$. If the

queries against these objects request the combinations (A_1, A_2, A_5) and (A_3, A_4) more frequently than any other combination, then it is desirable to have A_1, A_2 and A_5 placed in one class, and A_3 and A_4 placed in another. We refer to this problem as the Object Partitioning Problem (OPP). Indeed, to render the problem non-trivial, we assume that the joint access probabilities between the objects are unknown.

Before we proceed, we note that the basic assumption for this study is that there is an unknown underlying partition of the objects. This, in a sense, is determined by the user's query patterns. We shall refer to this partitioning of the objects as the **grouping** imposed on them. Observe that although this grouping is unknown, our aim is to partition the objects into classes that **mimic** the underlying **grouping**. For the rest of the thesis we shall use the words class and partition interchangeably.

In fact, the Object Partitioning Problem has been known to be NP-hard [27] due to the exponential growth in the number of partitions of the objects. For example, consider a special case when the underlying grouping consists of R **groups**, each containing exactly W/R objects. Obviously, we should partition the objects such that each **class** has the same number of objects. Although this restricts the number of classes to be considered, as we shall see, the number of possible partitions still grows exponentially as the number of objects increases. We demonstrate this phenomenon by an example in which, for various values of W , W/R is maintained at 5. In Table 1.1, we can see that when W is relatively small, say 10, the total number of partitions is only of order 10^2 . However, when W increases to 25, the number of partitions increases to the order of 10^{12} . This indicates an increase in the number of partitions of order 10^{10} , even though the number of objects has not even tripled.

No. Of Objects	Number of Possible Partitions of Size 5	
5	1	<div style="display: flex; align-items: center; justify-content: center;"> <div style="border-left: 1px solid black; height: 200px; margin-right: 10px;"></div> <div style="text-align: center;"> Grow Exponentially </div> </div>
10	126	
15	126126	
20	488864376	
25	$\sim 5.19 \times 10^{12}$	
30	$\sim 1.23 \times 10^{17}$	
.	.	
.	.	↓

Table 1.1 Exponential Growth in the Number of Partitions

1.2 Applications of the Object Partitioning Problem

There are various applications for the Object Partitioning Problem (OPP). For example, in a library environment [22], it is preferred to have books in related subjects located close together. In this case the set A could represent the set of subjects (e.g. history, geography, biology, electronics, etc.), and the partitioning would represent the physical location of the books and documents in these individual subjects (i.e., how these individual subjects should be grouped together). Thus, assuming that a person working in electronics required books in mathematics more frequently than books in geography, it is more desirable that books in electronics and mathematics be located in closer proximity than books in electronics and geography. Thus, in this context, the

solution of the OPP can be applied to handle the problem of subject clustering.

We present below some other computer science applications of the OPP:

- a) Suppose A is a set of attributes belonging to a relation in the third (or fourth) normal form. It may be that these attributes cannot be physically incorporated in one single relation because of the size of A , or in order to reduce the amount of data to be accessed per query, it is often desirable that these attributes are stored in different sub-relations. In this case, one possible criterion for subdividing the relation would be to do it in such a way that the attributes accessed simultaneously in a query are located in the same sub-relation [4,5,27]. Indeed, this is a version of the familiar Attribute Partitioning Problem.
- b) Let A be a set of records. In order to minimize the number of blocks needed to be accessed, it is required that these records be partitioned so that the jointly accessed records are stored in the same block (or track of the disk). The solution of the OPP can directly be applied to solve this problem.
- c) In distributed data bases, the files that are distributed can be located in various sites. It is desirable that files that are jointly accessed be physically located in the same geographical site so that communication costs can be minimized [11,27]. The solution of the OPP can again be used to handle this problem.

Since there are various applications of the OPP, we desire that solutions to it should be general, and that with minor modifications, these solutions should lend themselves to any of the above mentioned problem.

1.3 A Related Problem in the Literature

As mentioned in [27], a closely related problem that has been studied extensively is the one of organizing a linked list of N elements $\{R_1, \dots, R_N\}$ [1-3,6,7,12,18-21]. The aim is to organize these elements in the descending order of their (unknown) access probabilities s_i , $1 \leq i \leq N$. Of course, one can organize these individual elements by keeping a count on the number of times they are accessed, which is equivalent to estimating their access probabilities. However, since this required excessive memory, the search for self-organizing algorithms was motivated. Due to the similarity of the solution to the OPP, and various list organizing strategies, we shall briefly describe some algorithms for the latter problem.

McCabe [8,12] was the first to propose a solution to this problem. His solution is a dynamic self-organizing scheme and involves moving an element to the front of the list every time it is accessed. This scheme is called the Move-To-Front rule. Many researchers [3,6,7,20,21] have also extensively studied this rule and various properties of its limiting convergence characteristics are available in the literature.

McCabe [8,12] also introduced a scheme called the transposition rule. In this rule, whenever an element is accessed, it is interchanged with its preceding element unless it is at the front of the list. Much literature is available on the transposition rule [1-3,8,12,20] but particularly important is the work of Rivest [20] and Tenenbaum *et al* [1]. They studied the transposition rule extensively and suggested generalizations of it — the Move- K -Ahead rule and the POS(k) rule. In the former rule, an accessed element is moved k positions forward unless it is found in the first k positions. In such a case, it is just moved to the front of the list. In the latter rule, the accessed element is moved to

position k if it is in positions $k+1$ through n . It is transposed with its preceding element if it is in positions 2 through k and is left unchanged if it is the first element.

Recently, Oommen and Hansen [19] proposed two learning algorithms in which the elements of the list "learn" to find their positions adaptively. The first algorithm in [19] is essentially a move-to-front scheme with the exception that on the n^{th} access the element being accessed is moved to the front of the list with a probability $f(n)$. This probability is systematically decreased every time an element is accessed in such a way that the scheme is both absorbing and expedient, i.e. if $s_i > s_j$, then the probability of absorbing into an arrangement where R_i precedes R_j is always greater than 0.5.

The second algorithm in [19] is far more powerful. This algorithm is a move-to-rear scheme in which the accessed element R_i is moved to the rear of the list with a probability q_i . The quantity q_i is progressively decremented every time R_i is accessed. Hence, this scheme is also absorbing and could converge into any one of its $N!$ arrangements. Besides, it has been shown that the probability of converging to the optimal arrangement can be made as close to unity as desired. The algorithms in [19] are the only two known absorbing techniques discussed in the literature.

By considering the work done to solve the list organizing problem, one would hope to extend one of these solutions to yield an adaptive solution to the OPP. Also, we would like the solution to avoid estimating the access probabilities of the various combinations of the objects. Unfortunately, the solutions for the list organizing problem are not directly usable because the assumption that objects are accessed independently is quite meaningless in the context of the OPP. Further, the question of physically moving objects after

every access seems quite unreasonable, especially in the case of attribute partitioning or distributed file allocation. Thus, the need for a substantially different scheme is obvious, namely a scheme which considers the particular intricacies of the problem at hand.

1.4 Previous Solutions and Our Proposed Strategy

Various solutions for the OPP are discussed in the literature. The most obvious method to solve this problem is to process the queries and to maintain a frequency count of the accesses for various combinations of the objects. Since the number of combinations is usually very high, the use of such statistics are discouraged on the basis of both time and storage requirements.

The best known adaptive solution available is probably the one proposed by Yu *et al* [27]. Its closest competitor is possibly the technique by Hammer *et al* [4]. The latter one is a clever algorithm, which can be thought of as a "hill-climbing" technique that tries to converge to the optimal partitioning by using a heuristic approach. At its time it probably was the best algorithm to solve the attribute partitioning problem. Later, in [27], Yu *et al* proposed an adaptive scheme which they have called the Basic Adaptive Method (BAM).

The heart of the OPP is to find a partition of a given set of objects based on some of the user's queries. Hammer *et al* considered this as an optimization problem in which an optimal (or a near-optimal) solution was obtained by considering various possible partitions of the objects and finding the best partition among them. On the other hand, in the Basic Adaptive Method (BAM), Yu *et al* considered it as a clustering problem and the well-established theory of clustering was applied to partition objects into their respective clusters. Based on the clusters obtained, the objects were partitioned accordingly.

In this thesis, however, we employ a completely different approach to handle the OPP. In fact, we view the problem as a **learning problem** and apply the concepts of learning automata to solve it. Further, we show that the learning automata approach can be refined into two sub-approaches which correspond to Hammer's approach and Yu's approach, respectively.

In the first approach, no object needs to participate in the learning process; only a learning automaton is involved. The automaton employs the trial and error strategy and uses the information obtained from the environment (the query system) which it interacts with, to locate a suitable partition among all the possible partitions available to it. On the other hand, in the second approach, all the given objects are required to participate in the learning process. During the learning process, these objects choose actions from the available action set and move around in the automaton according to the information received from the environment with which they interact. The aim of the interaction of the **objects** is to find a suitable partitioning between the objects themselves, which hopefully is a solution to the OPP.

In this thesis, these two approaches are applied to tackle the OPP. We shall search for solutions to the OPP by either applying existing learning automata or by constructing new automata.

1.5 Thesis Outline

Chapter Two of this thesis briefly presents the "hill-climbing" technique for solving the Attribute Partitioning Problem proposed by Hammer *et al.* It also discusses the Basic Adaptive Method (BAM) due to Yu *et al.* In Chapter Three we discuss a brief overview of the area of learning automata. In this chapter, the concept of learning and the fundamentals of both fixed structure stochastic

automata and variable structure stochastic automata are discussed. Chapter Four initiates the discussion of our solutions to the OPP by first considering a special case of the problem, namely the Equal Partitioning Problem (EPP). In this case, we constrain the OPP by requiring that the objects are partitioned into classes of equal size. Two well-known deterministic automata, the Tsetlin and the Krinsky, are applied to the EPP. Although these two automata are ϵ -optimal, they are feasible only when the object size is small. Also introduced in this chapter is a new automaton which we have called the Object Migrating Automaton (OMA). The characteristics of this automaton are discussed in a fairly detailed manner. Unlike the Tsetlin and the Krinsky automata, abstract objects, which represent the actual physical objects, are partitioned by moving them around in the OMA. To avoid the situation of having several classes of objects coalescing into the same group, a limit is put on the size of each group. Experimentally, this automaton demonstrates expediency and ϵ -optimality. The difficulties encountered in generalizing the three automata models used in Chapter Four to handle the OPP are discussed in Chapter Five. Subsequently, another automaton, called the Stochastic Move Automaton (SMA), is developed and studied. This automaton renders to the abstract objects which move around in it the choice of stochastically changing actions when a change may be required. Although the SMA does not provide satisfactory results in solving the OPP, it is certainly a stepping stone to the development of a new variable structure automaton. In Chapter Six, this new variable structure automaton is used to tackle the OPP. This automaton is called the Modified Linear Reward-Penalty (ML_{RP}) automaton and employs an absorbing updating scheme derived from the well-known Linear Reward-Penalty scheme. This solution demonstrates a remarkably good accuracy. In every chapter, the

experimental results we obtained are compared with the results obtained by the best technique known in the literature, namely the Basic Adaptive Method (BAM) by Yu *et al.* Finally, Chapter Seven concludes the thesis and suggests the areas for further research.

Throughout this thesis, we assume the knowledge of the elementary concepts of relational data bases. The reader is referred to [28] for a reference.

Chapter 2

Previous Work

The Object Partitioning Problem (OPP) has been studied by many researchers and different types of solutions have been proposed. The simplest solution, of course, is the one which computes estimates of the query statistics (i.e., classifies queries into different types and effectively maintains a frequency count for each type). The objects are then partitioned based on the frequencies of the various types of queries. However, as mentioned in Chapter One, this method is, in general, not encouraged due to the exponential growth of the possible query types with the number of objects.

Various other solutions have been discussed in the literature. Most of them use heuristics and are not supported by any rigorous mathematical analysis. Since the QPP is NP-hard, any algorithm that gives an optimal solution will run in exponential time [27]. Thus, if experimental results are used to verify the correctness of these heuristics, the interpretation of these results will be limited. The "hill-climbing" technique proposed by Hammer *et al* is an example of one such algorithm. This technique has been verified only for object sets whose size is as large as eight because the simulation becomes computationally prohibitive if the number of objects is larger [27].

The best known algorithm available for the OPP probably is the Basic Adaptive Method (BAM) suggested by Yu *et al* [27]. Its advantages over other techniques are well defended in [27], its primary competitor being the technique of Hammer *et al* [4].

In the following sections, we shall first briefly describe the technique proposed by Hammer *et al* for the Attribute Partitioning Problem and then study

the BAM in the set up where exactly two objects are accessed per query. The case when more than two objects are accessed has also been handled in [27]. This preview will help us view the OPP in the right perspective and will help set the stage for the techniques which we propose.

2.1 A "Hill-Climbing" Approach to Attribute Partitioning

Hammer *et al* proposed in [4] a technique which uses the idea of "hill-climbing" (stepwise minimization) to search through the space of all possible partitions of a given set of attributes and locate a near-optimal partition. There are two main components used in the heuristic, namely a **Partition Evaluator (PE)** and a **Heuristic Searcher (HS)**. The purpose of the PE is to compute a figure of merit for any given partition of a set of attributes based on its physical characteristics and its usage patterns. The PE achieves its goal by using a **Transaction Cost Estimator (TCE)** which finds out the cost of processing any given transaction in the context of the given partition. The use of the HS is to conduct a search through the space of all possible partitions by using the "hill-climbing" technique and to locate a near-optimal partition based on the user's transactions.

2.1.1 The Method

The heuristic technique used in [4] is composed of two phases, namely the **Pairwise Grouping** and the **Attribute Regrouping**. These two phases are applied alternately until no improvement can be obtained. Equivalently, the phases are applied alternately until the cost of processing the transactions cannot be reduced by forming variations of the proposed partition. Suppose R is the relation [28] to be partitioned and is composed of m attributes. Then a

partition of $\mathcal{R}$ is defined to be a set of blocks where each block contains a subset of the attributes of $\mathcal{R}$. A candidate for the heuristic is therefore a valid partition of the attributes of $\mathcal{R}$. The initial candidate for the heuristic is the trivial partition in which all the m attributes are in separate blocks.

The first phase is the pairwise grouping phase. At each step of this phase, a set of variations is constructed by considering all pairwise groupings of the blocks of the current candidate. Thus, ${}_m C_2$ variations (where ${}_A C_B$ is the binomial coefficient having the value $A! / (B! (A-B)!)$) will be considered in the first step, for there are m attributes in $\mathcal{R}$. The set of variations are then sent to the PE for cost evaluation, and the one which has the greatest improvement over the current candidate becomes the new current candidate. This phase will continue until a state is attained from which no improvement can be obtained.

The second phase is the attribute regrouping phase which begins with the final candidate of the first phase. At each step of this phase, variations of the current candidate are constructed by moving an attribute from its current block to another block in the partition. Each variation entails the movement of only one attribute, and hence $m * (k - 1)$ variations need to be considered, where k is the number of blocks in the current candidate. Just like the first phase, the variations are sent to the PE for cost evaluation, and the one that shows the greatest improvement over the current candidate becomes the new current candidate. As before, this phase will continue until a state is attained from which no further improvement over the current candidate is obtainable.

The two phases of the heuristic are applied alternately until neither of them can yield any improvement over the current candidate. At this point the procedure terminates, and the current candidate is suggested as the "nearly-optimal" partition.

2.1.2 Intuition of the Heuristic

The intuition behind the pairwise grouping is as follows : Since all the m attributes of $\mathcal{R}$ are initially placed in separate blocks, those transactions that request only two attributes are answered with close to minimal cost while those transactions that request k attributes, $2 < k \leq m$, are more expensive to process. As the attributes cluster together, the processing of those transactions which use only two attributes becomes more expensive, primarily because unnecessary information from the other attributes found in the same block will also have to be processed. On the other hand, those transactions that use many attributes become computationally less expensive because most (or all) of the requested attributes are located in the same block. Eventually, an equilibrium point will be reached. At this point the increase in cost of processing those transactions that are expensive due to the grouping will not exceed the reduction in cost of processing the transactions that are benefited by the grouping.

The idea of the attribute regrouping is that the grouping obtained in the first phase may turn out to be faulty in the context of subsequent queries. By regrouping the attributes, the partition obtained in the first phase may be improved further. Hence, the first phase is invoked again, and the algorithm continues.

2.1.3 Experimental Results

Simulating the technique described above involves building up a database management system and designing methods for estimating costs in various transaction processing situations. Since this in itself is a mammoth undertaking, such a simulation was not attempted. However, the paper by

Hammer *et al* does provide some experimental results of the heuristic, and we would refer the reader to [4] for a review of their results and the details of the heuristic. Further, since this method is not as good as the Basic Adaptive Method suggested by Yu *et al*, we believe that a more meaningful comparison can be obtained by critically comparing any new scheme with the latter method. Thus, to have a better understanding of the latter method, for the rest of this chapter, we shall discuss the Basic Adaptive Method by Yu *et al* [27].

2.2 The Basic Adaptive Method

The Basic Adaptive Method (BAM) is intuitively simple, and is based on the idea of **clustering** the objects to be partitioned into classes based on the sequence of queries against them. As discussed in [27], the BAM possesses several advantages over many other solutions to the OPP. Some of these advantages are :

- (a) The scheme uses a unifying approach. Thus, the BAM can be applied to many other problems such as attribute partitioning, record clustering, *etc.*, with minor modifications.
- (b) The scheme does not involve collecting any query statistics.
- (c) The method is adaptive and is conceptually very simple.
- (d) The scheme has been mathematically analyzed.

We believe that the BAM is the best known algorithm available for the OPP. Although we shall present below a fairly comprehensive description of the BAM, we refer the reader to [27] for a more detailed exposition of the finer points.

2.2.1 Abstract Objects

As opposed to list organization schemes, on processing a query, the BAM does not move the given physical objects. Instead, the concept of **abstract objects** is introduced in [27] and the idea is as follows : Associated with the set of physical objects $\mathcal{A} = \{A_1, \dots, A_W\}$ is a set of **abstract objects** $\{O_1, \dots, O_W\}$. The BAM only manipulates **these abstract objects** as the queries are processed, and these abstract objects are partitioned into classes periodically. If the partitioning of these abstract objects are different from that of the corresponding physical objects, then the set of physical objects $\mathcal{A}$ will be re-partitioned so as to conform to the partitioning dictated by the BAM. The periodical repartitioning of $\mathcal{A}$ can be done when queries on the actual physical objects are not expected. Unlike the physical objects which usually consist of a lot of information, each abstract object merely contains its name and, in the case of the BAM, a position on the real line. Since only abstract objects are manipulated and they usually contain relatively little content, the cost of manipulation is, in general, greatly reduced. Further, as mentioned in Section 1.3, moving the physical objects in the case of attribute partitioning or distributed file allocation as queries are processed is unreasonable, and often impossible. However, such problems are eliminated automatically when the notion of abstract objects is used.

2.2.2 Convention

Throughout this thesis, we shall distinguish physical and abstract objects by the symbols A and O , respectively. Thus, A_i denotes the i^{th} **physical object** in a given set of physical objects, and O_i is used to represent the i^{th} **abstract object** in the corresponding set of abstract objects. Although all the queries will

only access physical objects, all the subsequent algorithms discussed in this thesis will only manipulate abstract objects. Further, when a set of physical objects is given, this automatically implies the existence of the corresponding set of abstract objects and vice versa.

2.2.3 The BAM Algorithm

Suppose a set of physical objects $\{A_1, \dots, A_W\}$ is given. Queries are issued against this set of objects such that only two objects are accessed in each query. Initially, the BAM assigns each abstract object O_i a real number X_i representing the position of that object on the real line, for all $1 \leq i \leq W$. On processing a query requesting the physical objects A_i and A_j , the corresponding abstract objects O_i and O_j are conceptually moved toward each other by moving the quantities X_i and X_j toward their centroid by a small constant Δ_1 . The centroid of the two quantities X_i and X_j is defined to be $(X_i + X_j) / 2$. This tends to bring the accessed objects into the same cluster. However, if this step is repeated for a long time, then all clusters corresponding to different classes may coalesce. In order to avoid this undesirable occurrence, two distinct random indices p and q , $1 \leq p, q \leq W$, representing two distinct objects O_p and O_q , are subsequently chosen, and they are moved apart from each other by moving the corresponding quantities X_p and X_q away from their centroid by a constant Δ_2 . The techniques of "determining" Δ_1 , Δ_2 and the initial X_i 's are informally discussed in [27]. If no *a priori* information about the objects is available, the values of both Δ_1 and Δ_2 can simply be set to unity.

The above two steps are executed for each query. Since the first step moves the accessed objects together, and the second step randomly pulls objects apart, only frequently accessed pairs of objects will eventually stay

together. After a fairly long time (T accesses), the partition is created based on the proximity of the X_i 's and is suggested as a solution to the OPP. For the sake of completeness, we present below an algorithmic version of the BAM.

Algorithm BAM

Input : The abstract objects $\{O_1, \dots, O_w\}$, two parameters Δ_1 and Δ_2 , and the stream of queries. Every query is conceptually a pair (A_i, A_j) .

Output : A periodic clustering of the objects into R partitions.

Method : Initialize X_i arbitrarily, where $-\infty < X_i < \infty$, for all $1 \leq i \leq W$.

Repeat

For a sequence of T queries **do**

 ReadQuery (A_i, A_j)

If $(X_i < X_j)$ **then**

$X_i = X_i + \Delta_1$

$X_j = X_j - \Delta_1$

Else

$X_i = X_i - \Delta_1$

$X_j = X_j + \Delta_1$

EndIf

 Select randomly distinct indices p, q , where $1 \leq p, q \leq W$

If $(X_p < X_q)$ **Then**

$X_p = X_p - \Delta_2$

$X_q = X_q + \Delta_2$

Else

$X_p = X_p + \Delta_2$

$X_q = X_q - \Delta_2$

EndIf

EndFor

 Print out partitions based on proximity of $\{X_i\}$

Forever

End Algorithm BAM

2.2.4 An Illustrative Example

We would like to illustrate how the BAM works by considering an example of partitioning a set of four objects $\{A_1, A_2, A_3, A_4\}$ into two classes. Suppose queries are issued against these objects in a way such that the pairs (A_1, A_2) and (A_3, A_4) are accessed more frequently than any other pairs. Initially, each abstract object O_i , $1 \leq i \leq 4$, is assigned to a location X_i on the real line. For simplicity, we assume that all the X_i 's have integral values and the values of both Δ_1 and Δ_2 are unity. Figures 2.1 and 2.2 illustrate the idea of the BAM. For the sake of simplicity, only the indices of the objects are specified (i.e., the index i represents the object O_i).

In this case, if only the first step of the algorithm is used for each query, (i.e., moving of accessed objects toward their centroid), then two clusters, with objects A_1, A_2 in one cluster and objects A_3, A_4 in the other, will be formed after a few steps. However, these two clusters are so closed together that they can easily be treated as one "big" cluster. See Figure 2.1.

On the other hand, if the second step is also used, namely the one which randomly moves a pair of objects away from their centroid, then only two well-defined clusters emerge. In this case, these two clusters are quite distinct from each other, as seen in Figure 2.2. A correct partition is thus obtained.

Figure 2.1 An Example of using only the first step of the BAM.

Figure 2.2 An Example of the BAM

Notation : (x, y) -- move objects 'x' and 'y' toward their centroid by Δ_1 .
 {x, y} -- move objects 'x' and 'y' away from their centroid by Δ_2 .

2.2.5 Between-Class and Within-Class Distances

When the BAM is applied to partition a set of W objects, the objects belonging to the same class will move closer to each other while objects belonging to different classes will move away from each other. Eventually, a set of classes (clusters) will be formed. In order to decide whether a given set of clusters is reasonable, two quantities, namely the **within-class distance** and the **between-class distance** are introduced. These quantities aid the BAM in its decision making.

Consider the special case when the W objects are partitioned into two classes. Suppose the positions of the objects in the first class and in the second class are $\{x_1, \dots, x_m\}$ and $\{y_1, \dots, y_t\}$, respectively, where $m + t = W$. Then, as defined in [27], the **within-class distance** of the objects is

$$W = \sum_{1 \leq i \leq m} |x_i - x'| + \sum_{1 \leq j \leq t} |y_j - y'|, \quad (2.1)$$

where

$$x' = \sum_{1 \leq i \leq m} x_i / m \quad \text{and} \quad y' = \sum_{1 \leq j \leq t} y_j / t \quad (2.2)$$

x' and y' are the centroids of the first class and the second class of the objects, respectively. The **between-class distance** of the objects in the two different classes is defined to be $B = |x' - y'|$. If we denote the change in the within-class distance as ΔW and the change in the between-class distance as ΔB , then the expected value of ΔW , $E[\Delta W]$, can be shown to be negative. Further, the expected value of ΔB , $E[\Delta B]$, can be shown to be positive. This implies that the objects in the same class will move closer to each other while the objects in different classes will move farther away from each other.

In general, if there are k classes, and if object i in class j has the position

x_{ij} , then we can define the within-class distance of the objects as

$$W = \sum_{1 \leq i \leq k} \sum_{1 \leq j \leq t} |x_{ij} - x_i'|, \quad (2.3)$$

where t is the number of objects in class i and x_i' represents the centroid of the objects in class i . The between-class distance is defined as

$$B = \sum_{1 \leq i \leq k} \sum_{1 \leq j \leq k \text{ and } j > i} |x_i' - x_j'| \quad (2.4)$$

2.2.6 Cluster Identification

Since the BAM discussed in Section 2.2.3 does not identify classes (clusters), the following method is presented in [27] which first identifies a set of potential classes and then decides whether this set is reasonable.

- (a) The first step is to identify a set of potential classes. A **potential class** is a set of objects in which the distance between an object and its right hand side neighbour, called the neighbouring distance, is less than a predefined quantity. This step is carried out for all objects on the real line to identify all the potential classes.
- (b) After the BAM has been invoked for a period of time, a set of potential classes P_1 is identified. The BAM is then invoked for some additional amount of time and another set of potential classes P_2 is identified. If
 - (i) the two sets of potential classes P_1 and P_2 are identical, and
 - (ii) if ΔB (where ΔB is defined to be the B due to P_2 minus the B due to P_1) is **positive** and ΔW (the W due to P_2 minus the W due to P_1) is **negative**, then the set of potential classes (either P_1 or P_2) is assumed to be reasonable.

Otherwise, the BAM will continue and another set of potential classes will be identified. This method terminates when the above two conditions are satisfied for two **consecutive** sets of potential classes.

2.2.7 Experiments Conducted

The BAM was simulated to obtain experimental results so that comparisons can be made between the BAM and the learning automata solutions that are discussed in later chapters. The objectives of the experiments were primarily to :

- (a) obtain the rate of convergence and the accuracy as a function of the number of objects, when the objects are partitioned into equal size classes.
- (b) obtain the rate of convergence and the accuracy as a function of the number of objects, when objects are partitioned into classes, not necessarily of equal size.

Various values of W (number of objects) and R (number of classes) were used. In each case, **one hundred** experiments were conducted in order to obtain accurate results. Besides, each experiment was allowed to process a maximum of **50,000** queries.

2.2.7.1 Initialization

The values of the X_i 's discussed in Section 2.2.3 were uniformly distributed in the interval $[0,100]$ initially because a very tight initial distribution of X_i 's could lead to the convergence involving a single class. The range $[0,100]$ was used as a standard in all the experiments for it caused the BAM to yield good partitioning results, especially when the number of objects was large.

2.2.7.2 Parameters

Three parameters were involved in the BAM, namely the Δ_i 's and the neighbouring distance (used in identifying potential classes). In all the experiments performed, these parameters were set up as follows :

- (a) Since no *a priori* information was given, Δ_1 and Δ_2 were set to **unity** as suggested in [27].
- (b) Although the X_i 's were initially distributed in the interval [0,100], they were spread throughout a fairly large interval after the BAM had been invoked for a while. In this case, if the distance between an object and its right hand side neighbour was set to a small value, then it would take quite a long time for the BAM to identify a reasonable set of potential classes. Therefore, we chose to set this parameter to be 50 for this value made the BAM identify a reasonable set of potential classes in a "reasonable" amount of time.

2.2.7.3 Assumptions

In all the experiments performed, queries were generated based on an underlying distribution unknown to the BAM algorithm and were based on the following assumptions :

- (a) The probability of accessing any pair of objects in one group is the same as that of accessing any other pair of objects in the same group.
- (b) The probability of accessing any pair of objects in different groups is the same as that of accessing any other pair of objects in different groups.
- (c) The probability of accessing a pair of objects in the same group is greater than that of accessing a pair of objects in different groups.

Thus, if π_i was the group in which the physical object A_i is located, then

these queries obeyed the following distribution:

$$\sum_{A_j \in \pi_i} \Pr [A_i, A_j \text{ accessed together}] = P, \quad i \neq j \quad (2.5)$$

Observe that if $P = 1$, then the partitioning is ideal (i.e., queries will involve only objects in the same partition). However, as the value of P decreases, the queries will be decreasingly informative about the underlying distribution governing the queries. Indeed, the value of P should be greater than 0.5 due to assumption (c).

2.2.7.4 Results

A summary of the results obtained is found in Table 2.1. The method of collecting potential classes described in Section 2.6 was used to obtain the partitioning of the objects. From the table, we observe that the BAM runs in exponential time as the number of objects increases. For example, when six objects were partitioned into three equal-size classes, the BAM converged to the expected partition after 269 iterations. However, when the number of objects increased to twelve, the corresponding figure is 2925. This implies an increase in time of convergence by a factor of about eleven when the size of objects is merely doubled. Further, the data for the case when eighteen objects were partitioned into two equal-size classes could not be obtained, primarily because it was too time consuming.

Apart from the rate of convergence, the BAM is in general quite accurate. In our report of the results, we shall say that an experiment "failed" if it did not yield the underlying partitioning of the objects after processing the 50,000 queries. In all the experiments performed, only two sets did not give 100 per cent accuracy. In the case where twelve objects were partitioned into four

unequal-size classes, four out of the hundred experiments performed failed to give the expected partition. In the case where these twelve objects were partitioned into two equal-size classes, nineteen out of the hundred experiments failed. Among these which failed, one of the expected classes was actually partitioned into two sub-classes. Such results are typical, and in agreement with the results reported in [27].

2.2.8 Complexity of the BAM

As mentioned in [27], the **time complexity** of the BAM is $O(T_{\text{BAM}})$, where T_{BAM} is the number of queries processed. The **space complexity** is $O(W)$, where W is the number of objects to be partitioned. Besides, $O(W \log W)$ steps are required to identify the objects into potential classes since this involves the sorting of these objects according to their positions on the real line.

No. Objects	Objects/Class	No. Classes	# iterations BAM	Accuracy
4	2	2	(21, 40)	100%
6	2	3	(255, 269)	100%
	2, 4	2	(55, 70)	100%
	3	2	(243, 251)	100%
9	3	3	(745, 762)	100%
	2, 3, 4	3	(365, 377)	100%
	4, 5	2	(711, 728)	100%
12	2	6	(1520, 1527)	100%
	2, 2, 2, 3, 3	5	(1940, 1977)	100%
	3	4	(1885, 1894)	100%
	2, 3, 3, 4	4	(8193, 8197)	96%
	4	3	(2914, 2925)	100%
	6	2	(3593, 3609)	81%
15	3	5	(2501, 2508)	100%
	2, 3, 3, 3, 4	5	(8224, 8281)	100%
	3, 4, 4, 4	4	(3380, 3507)	100%
	5	3	(6442, 6445)	100%
18	2	9	(3348, 3352)	100%
	3	6	(2910, 2917)	100%
	6	3	(12754, 12756)	100%
	9	2	Not Available	Not Available

Table 2.1 Rate of Convergence and Accuracy of the BAM

Notation : Number of iterations is given as a pair (x,y), where x is the average number of iterations for the partitioning to be obtained; y is the average number of iterations for the algorithm to converge to the partitioning.

2.3 Concluding Remarks

In this chapter, we have reviewed the "hill-climbing" technique for solving the Attributing Problem proposed by Hammer *et al.* The Basic Adaptive Method (BAM) proposed by Yu *et al.* has also been discussed. This method is considered as the best known algorithm available in the literature because of the advantages it possesses. Experimental results show that it provides excellent accuracy in solving the OPP.

Before we proceed to the discussion of solving the OPP by using learning automata, in the next chapter, we shall briefly review the fundamentals and the classification of various learning automata discussed in the literature.

Chapter 3

Overview of Learning Automata

Learning automata have been used to model biological learning systems and to find the optimal action which an environment offers. Historically, the first learning automata models were developed in mathematical psychology to explain behavioural patterns among living organisms [13]. The learning is achieved by actually interacting with an environment and processing the responses from the environment to the actions that are chosen. As time elapses, the learning system shows an improvement of its behaviour by moving toward an ultimate goal. Such automata have a variety of applications such as parameter optimization, statistical decision making and telephone routing [10,13–15]. Since the literature on learning automata is extensive, we refer the reader to a paper by Narendra and Thathachar [13] and a book by Lakshmivarahan [10] for an extensive review of the families and applications of learning automata. However, due to the fact that this thesis discusses different learning automata solutions to the OPP, we shall present briefly an overview of learning automata in the following sections.

3.1 The Concept of Learning

The process of learning, as described in [13], can be regarded as being similar to that of a student-teacher interaction. Suppose a question is posed to the student and a finite set of answers is provided. At each time instant, the student is required to choose an answer from the answer set and report it to the teacher. Should the teacher be deterministic, the problem is quite trivial. However, in many cases, the teacher's response is not necessarily

deterministic. Consider the case when the teacher responds to the student by stochastically rewarding or penalizing him. By using the response obtained from the teacher, the student updates his knowledge of whether the chosen answer is correct or not. Thus, a reward from the teacher implies that the chosen answer may be correct while a penalty implies that the chosen answer may not be correct. If the correct answer has the least penalty probability, one hopes that an intelligent student should be able to learn this answer. Indeed, many "intelligent organisms" (such as dogs and rats) are capable of such a learning behaviour. It is this learning characteristic which an automaton tries to model.

3.2 Learning Automata Models

The learning automata models discussed in the literature involve interactions between automata and random environments. By using the analogy of the student-teacher pair, a learning automaton can be regarded as the student and the random environment can be regarded as the teacher. As shown in Figure 3.1, the interaction between the automaton and the environment can be described as follows:

The automaton is offered a set of actions $\{\alpha_1, \dots, \alpha_R\}$ by the environment it interacts with. It is constrained to choose only one of these actions at a particular time. Once the automaton decides on an action, say α_i , this action will serve as an input to the environment. The environment will then respond to the input by giving out either a reward (the value '0') or a penalty (the value '1') based on the penalty probability c_i associated with α_i . This response in turn serves as an input to the automaton. Upon receiving the response from the environment, the automaton decides on its next action and the process repeats.

In general, a learning automaton is one which learns the optimal action (i.e., the action which has the minimum penalty probability), and eventually chooses this action more frequently than any other action.

Figure 3.1 The Automaton-Environment Interaction

Notation :

- (a) $\{ \alpha_1, \dots, \alpha_R \}$ is the set of actions, and α is the input to the environment.
- (b) $\{ c_1, \dots, c_R \}$ is the set of penalty probabilities, unknown to the automaton.
- (c) $\{ 0, 1 \}$ is the set of outputs from the environment. The value '0' is a reward and the value '1' is a penalty.

3.3 Random Environments

Suppose α_i is the action chosen by the learning automaton at the n^{th} time instant and serves as an input to the environment with which the automaton interacts. Upon receiving the input, the environment sends out a response $\beta(n)$, of value either '0' or '1', which is the feedback response to the automaton. The environment penalizes the automaton (i.e., $\beta(n) = 1$) with probability c_i , where

$$c_i(n) = \Pr [\beta(n) = 1 \mid \alpha(n) = \alpha_i] \quad \text{for } i = 1 \text{ to } R$$

Thus, the characteristics of the environment are specified by the set of penalty probabilities $\{c_1(n), \dots, c_R(n)\}$. If the c_i 's are **time invariant**, then the environment is said to be **stationary**; otherwise, it is said to be **non-stationary**. To make the problem non-trivial, it is assumed that the set of c_i 's are unknown to the learning automaton initially.

3.4 The Learning Objective

Under the assumption that the set of penalty probabilities $\{c_1, \dots, c_R\}$ is unknown initially, it is desired that as a result of interaction with the environment, the automaton arrives at the action which evokes the minimum penalty response in an expected sense. It may be noted that if c_L is the minimum penalty probability, and if we define

$$p_i(n) \triangleq \Pr [\alpha(n) = \alpha_i]$$

then

$$p_L(n) = 1, \quad p_i(n) = 0 \quad \text{for } i \neq L$$

achieves this result.

3.5 Learning Criteria

With no *a priori* information, the automaton chooses all the actions with equal probability. Thus, the expected penalty is initially M_0 , the arithmetic mean of the penalty probabilities. In other words,

$$M_0 = (c_1 + c_2 + \dots + c_R) / R$$

An automaton is said to learn **expediently** if, as time tends toward infinity, the expected penalty is less than M_0 . We denote the expected penalty at time 'n' as $E[M(n)]$. The automaton is said to be **optimal** if $E[M(n)]$ converges to the minimum penalty probability in the limit as time goes toward infinity. It is **ϵ -optimal** if in the limit $E[M(n)] < c_L + \epsilon$, where c_L is the minimum penalty probability, for any arbitrary $\epsilon > 0$. This could be achieved by a suitable choice of some parameter of the automaton, for example, the parameter of the updating scheme of the variable structure stochastic automata. In this case, the limiting value of $E[M(n)]$ can be as close to c_L as desired.

3.6 Classification of Learning Automata

The internal decision process of a learning automaton is modelled by using a transition matrix which describes the changes of the internal state of the automaton on receiving the responses of the environment. Similarly, the action it chooses is a function of its internal state, the function being determined by an output matrix. Based on the structures of the transition matrix and the output matrix, the automata discussed in the literature can be classified mainly into two categories :

- (a) Fixed Structure Stochastic Automata (FSSA)
- (b) Variable Structure Stochastic Automata (VSSA)

We now discuss these two categories in greater detail.

3.6.1 Fixed Structure Stochastic Automata (FSSA)

Fixed structure stochastic automata are those in which both the transition matrix and the output matrix are **time invariant**. Without loss of generality, the output matrix of an automaton can be deterministic [13]. Further, if the transition matrix is also deterministic, the automaton is usually called a **deterministic** automaton. The Tsetlin and the Krinsky automata are two well-known deterministic automata. In general, however, the transition matrix can be stochastic. The Krylov automaton is an example of such an automaton.

Formally, a FSSA is defined as a quintuple $(\alpha, \Phi, \beta, F, G)$, where

- (i) $\alpha = \{ \alpha_1, \dots, \alpha_R \}$ is the set of R actions offered by the environment that the automaton must choose from.
- (ii) $\Phi = \{ \phi_1, \dots, \phi_k \}$ is the set of states, with $k \geq R$.
- (iii) $\beta = \{ 0, 1 \}$ is the set of binary inputs from the environment. The input '0' represents a reward and the input '1' represents a penalty.
- (iv) F is the transition matrix. It is a map from $\Phi \times \beta$ to Φ and defines the transition of the internal state of the automaton on receiving an input. F can be either stochastic or deterministic.
- (v) G is the output matrix. It is a map from Φ to α and determines the action taken by the automaton if it is in a certain state. With no loss of generality, G can be deterministic [13].

3.6.1.1 The Learning Process of FSSA

In order to achieve the goal of learning, a learning automaton must be able to find out the optimal action from the action set and choose that action more frequently than any other action. In the case of FSSA, suppose a learning automaton is offered a set of R actions $\{ \alpha_1, \dots, \alpha_R \}$ and each action α_i has N

states $\{\phi_{i1}, \dots, \phi_{iN}\}$ associated with it, $1 \leq i \leq R$. With no loss of generality, we can assume that ϕ_{i1} and ϕ_{iN} are the most internal state and the boundary state of α_i , respectively. If at time 'n' the automaton chooses action α_i and in state ϕ_{ij} , where $1 \leq j \leq N$, and it receives a reward from the environment (i.e., $\beta(n) = 0$), then the automaton will interpret the reward as an indication that α_i may be the optimal action. Consequently, the automaton will go deeper into α_i by moving toward the most internal state of α_i , i.e. from ϕ_{ij} to ϕ_{ik} , where $1 \leq j, k \leq N$ and $k \leq j$. On the other hand, if a penalty is received from the environment (i.e., $\beta(n) = 1$), the automaton will assume that α_i may not be the optimal action and will start leaving it by moving toward the boundary state of α_i (i.e. from ϕ_{ij} to ϕ_{ik} , where $1 \leq j, k \leq N$ and $k \geq j$). If α_i is in fact the optimal action, then the automaton will receive rewards more frequently than if any other action were chosen. Thus, the automaton will ultimately circulate within the state near ϕ_{i1} , the most internal state of α_i . However, if α_i is not the optimal action, then the automaton will receive penalties more frequently than if the optimal action was chosen, and the automaton will eventually reach ϕ_{iN} , the boundary state of α_i , and will move to another action, say α_p , as specified by the transition map of the automaton. This procedure will repeat until the automaton arrives at the optimal action α_k and goes into it.

3.6.1.2 An FSSA Example

Consider the simple Tsetlin automaton shown in Figure 3.2 which consists of two actions $\{\alpha_1, \alpha_2\}$ and four states $\{\phi_1, \phi_2, \phi_3, \phi_4\}$ (i.e., each action has two states associated with it).

All transitions with probability 1.0

Figure 3.2 An FSSA Example: The Tsetlin Automaton

Using the notation described above, this automaton can be defined as a quintuple $(\alpha, \Phi, \beta, F, G)$, where

- (i) $\alpha = \{\alpha_1, \alpha_2\}$ is the set of actions that it must choose from
- (ii) $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$ is the set of states
- (iii) $\beta = \{0, 1\}$ is the set of inputs.
- (iv) F is the transition matrix from $\Phi \times \beta$ to Φ and is defined as follows :

$$\text{If } \beta = 0, \quad F_0 = \begin{matrix} & \begin{matrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{matrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{matrix} \end{matrix}$$

$$\text{If } \beta = 1 \quad F_1 = \begin{matrix} & \begin{matrix} \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{matrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{matrix} \end{matrix}$$

(v) G is the output matrix and is defined as follows :

$$G = \begin{matrix} & \begin{matrix} \alpha_1 & \alpha_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} & \begin{matrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{matrix} \end{matrix}$$

♦ ♦ ♦

3.6.2 Variable Structure Stochastic Automata (VSSA)

Although fixed structure stochastic automata are easy to implement, there is a major disadvantage with them. If an automaton is currently choosing an action, say α_i , then it will, almost always, choose the same action in the next time instant even if it receives a penalizing response from the environment. The only exception to this is when the automaton has reached the boundary state of α_i . In this case, the automaton will leave the current action and choose another one. This characteristic of the FSSA is not desirable when the number of states of an action is large for, indeed, this will make the automaton take a long time to leave any action, even if it is an undesirable one.

To overcome the drawback of the FSSA, the variable structure stochastic automata were introduced. Unlike the FSSA, the transition matrix and the

output matrix of the VSSA are **time variant**. Although the VSSA can be described in terms of transition and output matrices, in practice they are merely defined in terms of action probability updating schemes which are either of a continuous [9,10,13–15] or of a discrete nature [16,17] because of the following theorem proven by Varshavskii and Vorontsova [13, 26].

Theorem 3.1

For every stochastic automaton in which the transition and the output matrices are updated with time, there exists an equivalent updating scheme which involves **only** updating the action probability. ♦ ♦ ♦

The action probability vector $P(n)$ of an R -action automaton is defined as an R -vector $[p_1(n), \dots, p_R(n)]^T$, where $p_i(n)$ is the probability of choosing action α_i at time 'n' and the summation of all the p_i 's must be unity.

Since any VSSA chooses an action based on the distribution specified by its action probability vector, it is implemented by using a Random Number Generator (RNG). Due to the fact that almost all the VSSA discussed in the literature permit probabilities that can take any value between zero and one inclusive, by using a RNG that gives values in this interval the automaton can easily decide on the action to be chosen. Besides, since we use both a RNG and the action probability vector to decide on the action to be chosen, the VSSA has the added flexibility of choosing two **different** actions at two **consecutive** time instants. This feature indeed overcomes the drawback of the FSSA and therefore makes the VSSA, in general, converge much faster than the FSSA.

Action probability updating schemes studied in the literature are mainly divided into two mutually exclusive classes :

- (a) those which have absorbing barriers, and
- (b) those which are ergodic.

In the former class, the values of the limiting action probabilities **depend** on the initial distribution of the action probabilities. In the latter class, however, the distribution of the limiting action probabilities is **Independent** of the initial distribution of the values. In general, if the environment is stationary, schemes with absorbing barriers are more desirable. On the other hand, if the environment is non-stationary (i.e., the penalty probabilities are time varying), ergodic schemes are preferred because the automata will not get "locked" in any of the given actions. The Linear Reward-Penalty (L_{RP}) scheme discussed in the literature is an example of the ergodic scheme.

Formally, a VSSA is defined as a quadruple (α, P, β, F) , where

- (i) $\alpha = \{ \alpha_1, \dots, \alpha_R \}$ is the set of R actions offered by the environment that the automaton must choose from.
- (ii) $P = [p_1(n), \dots, p_R(n)]^T$ is the action probability vector with p_i representing the probability of choosing action α_i at the n^{th} time instant, $1 \leq i \leq R$. Also, $\sum_{1 \leq i \leq R} p_i(n) = 1$.
- (iii) $\beta = \{ 0, 1 \}$ is the set of binary inputs from the environment. The input '0' represents a reward and the input '1' represents a penalty.
- (iv) F is the updating scheme. It is a map from $P \times \beta$ to P and defines the method of updating the action probabilities on receiving an input from the environment. If $p_i(n)$ represents the probability of choosing action α_i at the n^{th} time instant, then F is usually of the form:

$$\begin{aligned}
 p_i(n+1) &= p_i(n) - g_i(P) && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) + \sum_{j \neq i} g_j(P) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
 p_i(n+1) &= p_i(n) + h_i(P) && \text{if } \alpha_j \text{ is chosen and } \beta = 1 \\
 p_i(n+1) &= p_i(n) - \sum_{j \neq i} h_j(P) && \text{if } \alpha_i \text{ is chosen and } \beta = 1
 \end{aligned}$$

where g and h are functions of P and $1 \leq i, j \leq R$ and $i \neq j$.

In general, if $P(n+1)$ is a linear function of the vector $P(n)$, then the updating scheme is said to be linear; otherwise, it is said to be non-linear.

3.6.2.1 The Learning Process of VSSA

Suppose at time 'n' the action probability distribution is $p_k(n)$, for $1 \leq k \leq R$, and the automaton decides to choose α_i . If $\beta(n) = 0$, then the automaton will go deeper into action α_i by increasing the value of p_i from $p_i(n)$ to $(p_i(n) + \Delta_1)$, $0 \leq \Delta_1 \leq (1 - p_i(n))$. Since $\sum p_i = 1$, for all $1 \leq i \leq R$, increasing the value of p_i implies decreasing the value of at least one p_j where $1 \leq j \leq R$ and $j \neq i$. On the other hand, if $\beta(n) = 1$, the automaton will decrease the probability of choosing α_i from $p_i(n)$ to $(p_i(n) - \Delta_2)$, $0 \leq \Delta_2 \leq p_i$, and will increase the value of at least one p_j where $1 \leq j \leq R$ and $j \neq i$. If α_i is in fact the optimal action, then whenever the automaton chooses α_i , it will be rewarded more frequently than if any other action were chosen. Eventually, it is hoped that the value of p_i will tend toward unity while the values of p_j , $1 \leq j \leq R$ and $j \neq i$, will tend toward zero. However, if α_i is not the optimal one, then whenever the automaton chooses α_i , it will be penalized more frequently than if the optimal action was chosen, and the value of p_i will be decreased. Eventually, the value of p_i will tend toward zero, and the value of p_k will tend toward unity, if α_k is the optimal action.

3.6.2.2 A VSSA Example

Consider a two-action variable structure learning automaton which employs linear updating functions for $g(\cdot)$ and $h(\cdot)$. By using the above described notation, this automaton can be defined as a quadruple (α, P, β, F) , where

- (i) $\alpha = \{\alpha_1, \alpha_2\}$ is the set of action that the automaton must choose from.
- (ii) $P(n) = [p_1(n), p_2(n)]^T$ is the action probability vector with $p_1(n)$ and $p_2(n)$ representing the probabilities of choosing α_1 and α_2 at the n^{th} time instant, respectively. Also $p_1(n) + p_2(n) = 1.0$. In future, unless explicitly stated, we shall omit the parameter n to simplify the notation. Thus p_i will represent $p_i(n)$.
- (iii) $\beta = \{0, 1\}$ is the set of binary inputs from the environment. The value '0' represents a reward and the value '1' represents a penalty.
- (iv) Suppose $g_i(P) = a * p_i$
 $h_i(P) = [b / (R-1)] - b * p_i,$

where $R = 2$ and $1 \leq i \leq 2$. In this case, F , the updating scheme, is defined as follows:

$$\begin{array}{lll}
 p_i(n+1) & = & p_i(n) - a * p_i & \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
 p_i(n+1) & = & p_i(n) + a * p_j & \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
 p_i(n+1) & = & p_i(n) + b * p_j & \text{if } \alpha_j \text{ is chosen and } \beta = 1 \\
 p_i(n+1) & = & (1 - b) * p_i(n) & \text{if } \alpha_i \text{ is chosen and } \beta = 1
 \end{array}$$

for $1 \leq i, j \leq 2$ and $i \neq j$

This is the well-known Linear Reward-Penalty (L_{RP}) scheme.

3.7 Concluding Remarks

In this chapter we have introduced the elementary concepts of the theory of learning automata. The automaton aims to learn the optimal action offered by the environment with which it interacts. We have briefly discussed the two families of learning automata discussed in the literature, namely the Fixed Structure Stochastic Automata (FSSA) and the Variable Structure Stochastic Automata (VSSA). The philosophy behind these families have been discussed and illustrative examples have been given. In general, VSSA are more flexible and converge faster than FSSA.

In the next chapter, we shall proceed to study the OPP by first proposing deterministic learning automata solutions to handle a special case of the problem, namely the case when all the partitions are of equal sizes. This problem is referred to as the Equal Partitioning Problem (EPP).

Chapter 4

The Equal Partitioning Problem

Before studying the Object Partitioning Problem (OPP), we would like to consider a special case called the Equal Partitioning Problem (EPP). Throughout this chapter we assume that the underlying distribution of the query pattern supports a grouping in which the objects are in equal-size groups.

4.1 Equal Partitioning Problem

Let $\mathcal{A} = \{A_1, \dots, A_W\}$ be a set of W objects. The intention is to partition the set $\mathcal{A}$ into R classes $\{P_1, \dots, P_R\}$ such that each class has **exactly** W/R objects. The aim is to have the partitioning identical to the unknown underlying grouping specified by the user's patterns. In order to make the problem meaningful, we assume that $W = M \cdot R$, for some positive integer M . Further, the objects are to be partitioned in such a way that those accessed more frequently together lie in the same class.

Although the EPP is a simplified case of the OPP, it is still NP-hard due to the exponential growth in the number of partitions of objects as the size of the object set increases. In fact, as we shall discuss in Section 4.3.8, the number of possible partitions is equal to $W! / [(M!)^R R!]$, where W is the number of objects, R is the number of classes and $M = W/R$. To illustrate the exponential growth, let us consider the case when $W = 6$ and $R = 3$. In this case it can be seen that there are only 15 possible partitions. However, when W increases to 12, there are 5775 possible partitions. Doubling the size of the object set results in an increase in the number of partitions by a factor of 385 times. Hence the exponential growth in the number of partitions is obvious.

For the rest of this chapter, we shall discuss three FSSA solutions to the EPP. The first two solutions which we propose involve the Tsetlin and the Krinsky learning automata. The third solution is a new automaton which we shall call the Object Migrating Automaton (OMA).

4.2 The Tsetlin and the Krinsky Automata

Tsetlin [24,25] who initiated the work on FSSA was the first one to design a machine which actually had expedient and even ε -optimal learning behaviour. It was initially called the Linear Tactic and had only two actions in it. In general, a K-action Tsetlin automaton, $L_{KN,K}$, has $K \cdot N$ states $\{\phi_1, \dots, \phi_{KN}\}$ and it chooses one of the K actions $\{\alpha_1, \dots, \alpha_K\}$ at each time instant. The variable N is called the "depth" of the automaton (i.e., how far the automaton can go in each action). $L_{KN,K}$ can be explicitly defined by using the notation described in Chapter Three as follows :

$$L_{KN,K} = (\{ \alpha_1, \dots, \alpha_K \}, \{ \phi_1, \dots, \phi_{KN} \}, \{ 0, 1 \}, L(\cdot, \cdot), G(\cdot))$$

The $L(\cdot, \cdot)$ map is **deterministic** and is described as follows:

- (1) If $\beta = 0$, (it gets a favourable response), the L map requires that the automaton go toward the most internal state (i.e., $\phi_{(j-1)N+1}$ if α_j is the action chosen), one step at a time, for all $1 \leq j \leq K$. If it is already in the most internal state, then it just stays there.
- (2) If $\beta = 1$ (it gets an unfavourable response), the automaton is required to move toward the boundary state (i.e., ϕ_{jN} if α_j is the action chosen), one step at a time. If it is in the boundary state of α_j , it will then move to the boundary state of the next action, for all $1 \leq j \leq K$.

$$\begin{aligned} \text{Thus, } L(\phi_i, 0) &= \phi_{i-1} && \text{if } (j-1)N + 1 < i \leq jN \\ &= \phi_j && \text{if } i = (j-1)N + 1 \end{aligned} \quad (4.1)$$

$$\begin{aligned}
 L(\phi_i, 1) &= \phi_{i+1} && \text{if } (j-1)N + 1 \leq i < jN \\
 &= \phi_{((j+1)N) \bmod (KN)} && \text{if } i = jN
 \end{aligned} \tag{4.2}$$

In (4.2), if $j = K-1$, then $((j+1)N) \bmod (KN)$ will be zero. However, this is just equivalent to KN . The transition map is drawn in Figure 4.1 for the case when $K = 2$.

The $G(\cdot)$ map is deterministic and is defined as

$$G(\phi_j) = \alpha_i \quad \text{if } (i-1)N + 1 \leq j \leq iN \tag{4.3}$$

Observe that this means that the automaton chooses α_1 if it is in any of the first N states, it chooses α_2 if it is in any of the states from ϕ_{N+1} to ϕ_{2N} , etc.. Thus, the states of the automaton are automatically partitioned into groups, each group representing an action.

Figure 4.1 $L_{2N,2}$, The Two-action $2N$ -state Tsetlin Automaton

We now state the theorem proven in [24].

Theorem 4.1

The $L_{KN,K}$ automaton is ϵ -optimal in all random environments in which the minimum penalty probability is less than 0.5 ♦ ♦ ♦

Following the introduction of the Tsetlin automaton, various scientists proposed other FSSA which had good learning characteristics. The Krinsky automaton was one of these automata. In general, a K-action Krinsky automaton, $Z_{KN,K}$, possessing KN states $\{\phi_1, \dots, \phi_{KN}\}$ and K actions $\{\alpha_1, \dots, \alpha_K\}$ is defined as follows :

$$Z_{KN,K} = (\{\alpha_1, \dots, \alpha_K\}, \{\phi_1, \dots, \phi_{KN}\}, \{0,1\}, Z(\cdot, \cdot), G(\cdot))$$

The G map is deterministic and is defined as in (4.3), exactly as the G map of the Tsetlin automaton. Therefore, the automaton chooses α_i if it is in any state ϕ_j , where $(i-1)N + 1 \leq j \leq iN$.

On the other hand, although the Z map is deterministic, it is not the same as the L map of the Tsetlin automaton and can be defined as follows :

- (1) If $\beta = 0$ (obtains a favourable response), the automaton is required to move **directly** to the most internal state corresponding to the action chosen (i.e., $\phi_{(j-1)N+1}$ for action α_j , $1 \leq j \leq K$).
- (2) If $\beta = 1$ (i.e., it obtains an unfavourable response), the movement of the automaton is exactly the same as that of the Tsetlin automaton. In this case it moves toward the boundary state of the chosen action one step at a time.

$$\text{Thus, } Z(\phi_i, 0) = \phi_{(j-1)N+1} \quad \text{if } (j-1)N + 1 \leq i \leq jN \quad (4.4)$$

$$Z(\phi_i, 1) = L(\phi_i, 1) \quad 1 \leq i \leq KN \quad (4.5)$$

The transition map of the Krinsky automaton is shown in Figure 4.2 for the case when $K = 2$.

Figure 4.2 $Z_{2N,2}$, The Two-action $2N$ -state Krinsky Automaton

As opposed to the Tsetlin automaton, the Krinsky automaton has the following property [25]:

Theorem 4.2

The Krinsky automaton is ϵ -optimal in all random environments. ♦ ♦ ♦

4.3 The Tsetlin and Krinsky Automata Solutions to the EPP

4.3.1 Approach

Suppose a set of W physical objects $\{A_1, \dots, A_W\}$ is given. Just like the BAM, only the set of abstract objects $\{O_1, \dots, O_W\}$ associated with the physical

objects is manipulated by the algorithm. By using the Tsetlin and the Krinsky automata, the solution of the EPP is straightforward. If the set of objects is partitioned into R classes with each class containing W/R objects, then each possible partition can be regarded as an action of the FSSA, and the objective of the automaton is to find out the action representing the "optimal" partition of the objects.

For example, suppose a set of four objects $\{A_1, A_2, A_3, A_4\}$ is given and is to be partitioned into two classes such that each class contains exactly two objects. Observe that this means that there are three possible partitions, namely $\{(A_1, A_2), (A_3, A_4)\}$, $\{(A_1, A_3), (A_2, A_4)\}$ and $\{(A_1, A_4), (A_2, A_3)\}$. These three partitions can be represented as the actions of a three-action Tsetlin or Krinsky automaton such that action α_1 represents the partition $\{(A_1, A_2), (A_3, A_4)\}$, action α_2 represents the partition $\{(A_1, A_3), (A_2, A_4)\}$, and so on.

4.3.2 Structures of the Tsetlin and the Krinsky Learning Automata

Having considered each action as a possible partition, the structures of the Tsetlin and the Krinsky automata are simple. If there are K possible ways to partition the given W objects into R equal-size classes, then a K -action automaton is needed in order to represent all the possible partitions. By using the notation defined in Section 3.6.1, such an automaton is defined as a quintuple $(\{\alpha_1, \dots, \alpha_K\}, \{\phi_1, \dots, \phi_{KN}\}, \{0, 1\}, L(\cdot, \cdot), G(\cdot))$, where

- (a) $\{\alpha_1, \dots, \alpha_K\}$ is the set of K actions that the automaton must choose from with each action representing a certain partition.
- (b) $\{\phi_1, \dots, \phi_{KN}\}$ is the set of states with $\phi_{(j-1)N+1} \dots \phi_{jN}$ representing the N states associated with action α_j , for all $1 \leq j \leq K$.
- (c) $\{0, 1\}$ is the set of inputs from the interacting environment. The values '0'

- and '1' represent a reward and a penalty, respectively. We shall, in a moment, describe what we mean by the environment, in this framework.
- (d) $L(\cdot, \cdot)$ is the transition matrix and is defined as in (4.1) and (4.2) for the case of the Tsetlin automaton and in (4.4) and (4.5) for the case of the Krinsky automaton.
- (e) The output matrix $G(\cdot)$ is the same as that defined in (4.3)

4.3.3 The Environment

The environment that the automaton interacts with is actually the "query system" (i.e., the set of user's queries). Once the automaton chooses an action, the environment gives out a reward or a penalty according to the following rules:

- (1) If the pair (A_i, A_j) is accessed and the action chosen by the automaton supports the partition that A_i and A_j should be placed together, then a reward (i.e., $\beta = 0$), will be given.
- (2) Conversely, if the query does not support the partition that A_i and A_j should be placed together, a penalty (i.e., $\beta = 1$), will be given.

4.3.4 Method

Initially, the automaton is assigned arbitrarily to an action α_p , where $\alpha_p \in \{\alpha_1, \dots, \alpha_K\}$. Suppose at the n^{th} time instant, a pair of objects (A_i, A_j) is accessed and the automaton is using action α_q , where $\alpha_q \in \{\alpha_1, \dots, \alpha_K\}$. If the query supports the partition represented by α_q , then the automaton will be rewarded and the movement will be defined by the transition map of the automaton. On the other hand, if the query contradicts the partition represented by α_q , the automaton will be penalized and will move toward the boundary state of the

chosen action according to its transition map.

Just like the BAM, after a period of T accesses, the physical objects are reorganized based on the partition represented by the current action of the automaton.

We shall present below algorithmic versions of the methods using the Tsetlin and the Krinsky automata. For simplicity, we assume the existence of the function "Support" which tells whether an input query supports a given partition or not.

Algorithm Tsetlin EPP Solution

Input : The number of actions K and the number of states N per action.
Also coming into the algorithm is a stream of queries (A_i, A_j) .

Assumption : Existence of function "Support".

Output : A periodic partitioning of the objects.

Notation : ξ is the current state of the automaton and is an integer in the range $1 \dots KN$.

Method : Initialize ξ arbitrarily such that $1 \leq \xi \leq KN$.

Repeat

For a sequence of T queries **do**

ReadQuery (A_i, A_j)

If (Support (A_i, A_j)) **then** (* Reward.*)

If $((\xi \bmod N) \neq 1)$ **then**

$\xi = \xi - 1$

EndIf

Else (* Penalty *)

If $((\xi \bmod N) = 0)$ **then**

$\xi = ((\xi / N + 1) \bmod K) * N$

If $(\xi = 0)$ **then**

$\xi = KN$

EndIf

Else

$\xi = \xi + 1$

EndIf

EndIf

Print out partitions based on the state ξ

ForEver

End Algorithm Tsetlin

Algorithm Krinsky EPP Solution

Input : The number of actions K , and the number of states N per action.
Also coming into the algorithm is a stream of queries (A_i, A_j) .

Assumption : Existence of function "Support".

Output : A periodic partitioning of the objects.

Notation : ξ is the current state of the automaton and is an integer in the range $1 \dots KN$.

Method : Initialize ξ arbitrarily such that $1 \leq \xi \leq KN$.

Repeat

For a sequence of T queries **do**

 ReadQuery (A_i, A_j)

If (Support (A_i, A_j)) **then** (* Reward *)

$\xi = \lfloor \xi / N \rfloor * N + 1$

Else (* Penalty *)

If $((\xi \bmod N) = 0)$ **then**

$\xi = ((\xi / N + 1) \bmod K) * N$

If $(\xi = 0)$ **then**

$\xi = KN$

EndIf

Else

$\xi = \xi + 1$

EndIf

EndIf

 Print out partitions based on the state ξ

ForEver

End Algorithm Krinsky

4.3.5 A Tsetlin or Krinsky Automaton Example

To illustrate how an optimal partition is obtained, let us consider an example of partitioning four objects $\{A_1, A_2, A_3, A_4\}$ into two equal-size classes (i.e., two objects per class). Based on the given information, three possible partitions, namely $\{(A_1, A_2), (A_3, A_4)\}$, $\{(A_1, A_3), (A_2, A_4)\}$ and $\{(A_1, A_4), (A_2, A_3)\}$ can be obtained. By using a $L_{6,3}$ Tsetlin or a $Z_{6,3}$ Krinsky automaton (i.e., the automaton has six states and three actions), the three partitions are represented by the three actions available to the automaton as follows :

$$\alpha_1 : \{(A_1, A_2), (A_3, A_4)\}$$

$$\alpha_2 : \{(A_1, A_3), (A_2, A_4)\}$$

$$\alpha_3 : \{(A_1, A_4), (A_2, A_3)\}$$

We have chosen to use only two states per action in this case since it will illustrate the behaviour of both the Tsetlin and the Krinsky automata. For simplicity, only the index 'i' is used to represent the object A_i in the illustrations.

Let us assume that the queries appear as below:

Query 1 : Access objects A_1 and A_2 .

Query 2 : Access objects A_1 and A_3 .

Query 3 : Access objects A_1 and A_2 .

Query 4 : Access objects A_3 and A_4 .

Step 0 : Initially, the automaton is assigned arbitrarily to choose α_2 , and be in the boundary state of that action (i.e., in state ϕ_4).

Step 1 : The pair (A_1, A_2) is accessed. Since the query does not support the partition represented by α_2 , the automaton is penalized by moving to the next action α_3 in state ϕ_6 .

Step 2 : The pair (A_1, A_3) is accessed. Again, the query does not support the partition represented by α_3 . The automaton is penalized and moves to α_1 in state ϕ_2 .

Step 3 : The pair (A_1, A_2) is accessed. Since α_1 represents the partition $\{(A_1, A_2), (A_3, A_4)\}$ and the query supports this partition, the automaton is rewarded by moving to ϕ_1 , the most internal state of α_1 .

Step 4 : The pair (A_3, A_4) is accessed. Again, the query supports the partition represented by α_1 and the automaton is rewarded. In this case, since the automaton is in the most internal state of α_1 , the algorithm is considered to have converged and the partition $\{(A_1, A_2), (A_3, A_4)\}$ is regarded as the solution to the EPP for the given objects.

4.3.6 Implementation of the Algorithms

The practical implementation of the learning machines described above entails the maintenance of one Integer memory location which stores the internal state of the automaton. Since the number of states per action is known, calculating the corresponding action of any given state can be done easily.

Apart from this, a static memory array must be allocated to "remember" the partitions represented by each action (i.e., by giving the action number, the corresponding partition can be obtained from the array, and vice versa). Observe that this static array can be precomputed, and further, need be computed exactly once.

4.3.7 Experiments

The EPP was simulated for two sets of objects with various probabilities of being jointly accessed. The Tsetlin automaton, the Krinsky automaton and the

BAM were each used to perform the learning of the optimal partitioning. The objectives of the experiments were primarily to:

- (a) study the rate of convergence and the accuracy of the methods when the objects are partitioned into equal-size classes,
- (b) study the rate of convergence and the accuracy of the methods as a function of joint access probability.

In case (a), two sets of simulations were performed. In the first set, W , the number of objects, was set to four and R , the number of classes, was set to two. In the second set, W and R were set to six and three, respectively. In case (b), the joint access probabilities, as defined in (2.5), used were in the range 0.6...0.9 with a step of size 0.1. In each set of simulations, one hundred experiments were performed in order to obtain accurate results.

4.3.7.1 Initialization

For the Tsetlin and the Krinsky automata, since *no a priori* information was given, the automaton was assigned arbitrarily to choose an action α_i and be in a state of that action, $\alpha_j \in \{\alpha_1, \dots, \alpha_K\}$. For the BAM, the initialization was the same as that described in Section 2.2.7.1.

4.3.7.2 Parameters and Assumptions

The parameter involved in the Tsetlin and the Krinsky automata was the number of states per action. In all the experiments performed, this value was set to 10. In the case of the BAM, the set up of the parameters was the same as that described in Section 2.2.7.2. The assumptions used were the same as that described in Section 2.2.7.3.

4.3.7.3 Convergence

To render the comparison between the algorithms meaningful, the Tsetlin and the Krinsky algorithms were said to have "converged" if the automata reached the most internal state of a particular action. This is quite realistic because when this occurs, the probability of leaving that action is very low. Further, this is comparable to the notion of convergence of the BAM discussed in [27].

4.3.7.4 Results

Table 4.1 summarizes the results that have been obtained. In this case, all the experiments converged to the optimal partition successfully (i.e., 100 per cent accuracy). From the table we observe that the automata solutions to the EPP are superior to the BAM. For example, in the case when $W = 6$ and $R = 3$, when $P = 0.8$, the BAM on the average takes 99 iterations to converge to the optimal solution. In this case the figure for the Tsetlin automaton is only 32 iterations. Similarly, the figure for the Krinsky automaton is 17 iterations. Notice that in this case, if we consider convergence as a criterion, the Tsetlin automaton is about 3:1 times faster than the BAM, and the Krinsky automaton is about 5.8 times faster than the BAM. Such results are typical.

No. Objects	No. Classes	P	# iterations BAM	# iterations Tsetlin	# iterations Krinsky
4	2	0.6	(40, 182)	(26, 56)	(44, 45)
		0.7	(29, 94)	(12, 31)	(26, 27)
		0.8	(22, 62)	(8, 23)	(15, 16)
		0.9	(18, 45)	(7, 18)	(9, 10)
6	3	0.6	(63, 513)	(49, 78)	(26, 27)
		0.7	(39, 153)	(24, 45)	(21, 22)
		0.8	(35, 99)	(17, 32)	(16, 17)
		0.9	(33, 73)	(14, 25)	(14, 15)

Table 4.1 Comparison of the BAM, the Tsetlin and the Krinsky automata solutions for the EPP.

Notation : P -- joint access probability defined by (2.5)

Number of iterations is given as a pair (x,y), where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning.

4.3.8 Comments

The Tsetlin and the Krinsky automata have good learning properties and are superior to the BAM. However, because of the following proposition, they are practically feasible only when W and R are small.

Proposition 4.1

For any particular value of W and R , the number of actions that a Tsetlin (or Krinsky) automaton solving the EPP must possess is $W! / [(M!)^R R!]$, where $M = W/R$.

Proof :

The number of actions that the automaton possesses must be equal to the number of equal-sized partitions that can be made from W objects into R classes. Let this number be $f(W, R)$. Trivially, this implies that in each class there will be M objects, since $M = W/R$.

Suppose the object A_1 is in any one class. Then, the number of distinct ways by which this class can be filled is ${}_{W-1}C_{M-1}$, where ${}_AC_B$ is the binomial coefficient having the value $A! / (B!(A-B)!)$. This leaves us with $R-1$ classes to be filled and $W-M$ objects with which to fill them. Thus, $f(W, R)$ obeys the difference equation :

$$f(W, R) = {}_{W-1}C_{M-1} \cdot f(W-M, R-1), \quad \text{where } M = W/R \quad (4.6)$$

Substituting

$$f(W-M, R-1) = {}_{W-M-1}C_{M-1} \cdot f(W-2M, R-2)$$

$$f(W-2M, R-2) = {}_{W-2M-1}C_{M-1} \cdot f(W-3M, R-3), \text{ etc.}$$

By simple induction, it can be seen that $f(W, R) = W! / (M!)^R R!$ satisfies (4.6) and the proposition is proved. ♦ ♦ ♦

Using the above proposition it is easy to see that if the number of objects is 12 and the number of classes is 3, the automaton must possess 5775 actions. Clearly, this renders these automata impractical, unless there is a straightforward way to code the actions into their partition representations such that on processing a query, a linear search will not have to be done.

4.3.9 Complexities of the Tsetlin and the Krinsky Automata

The time complexity of the Tsetlin automaton is $O(T_{\text{Tsetlin}})$, where T_{Tsetlin} is the number of queries processed. The space complexity is $O(S_{\text{Tsetlin}})$ where S_{Tsetlin} is the number of possible partitions (i.e., $W! / [(M!)^R R!]$).

Similarly, the time and the space complexities of the Krinsky automaton are $O(T_{\text{Krinsky}})$ and $O(S_{\text{Krinsky}})$, respectively, where T_{Krinsky} is the number of queries processed and S_{Krinsky} is the number of possible partitions.

Based on the experimental results that have been obtained, we conjecture that, in general,

$$\begin{aligned} O(T_{\text{Krinsky}}) &< O(T_{\text{Tsetlin}}) < O(T_{\text{BAM}}) \quad \text{and} \\ O(S_{\text{Krinsky}}) &= O(S_{\text{Tsetlin}}) > O(S_{\text{BAM}}) \end{aligned}$$

We shall now consider a FSSA which we have designed. This automaton is called the Object Migrating Automaton (OMA). This automaton is in general slower than the Tsetlin and the Krinsky automata, but it is still faster than the BAM and yet feasible when the number of objects to be partitioned is large.

4.4 The Object Migrating Automaton

4.4.1 Motivation

The EPP was first attempted by using an **R-action** Tsetlin automaton and the W given objects were initially assigned arbitrarily to the R actions such that each action had exactly W/R objects assigned to it. When a pair of physical objects (A_i, A_j) was accessed, and if the objects were using the same action, then they would be rewarded by moving toward the most internal state of the chosen action. On the other hand, if they chose different actions, then they would be penalized by moving toward the boundary states of the actions chosen and would move to the next action upon reaching the boundary state.

This method sounded reasonable until it was simulated and a serious problem was discovered. Since the method was simulated to partition a set of objects based on an underlying grouping unknown to the method, we were able to tell whether the method provided correct results or not. By studying the results that we had obtained, we discovered that this method was capable of grouping objects that were frequently jointly accessed together. However, it was **not** capable of preventing two or more groups of objects from coalescing into the same class. This was in fact a serious problem because the primary goal of the partitioning problem would be defeated if all objects entered the same class. This led us to the development of a new FSSA called the Object Migrating Automaton (OMA).

4.4.2 Approach

One reasonable way to prevent the objects from coalescing together is to put a "limit" on the size of each class such that each class has exactly enough room for one group of objects. Since the goal of the EPP is to partition the W

given objects into R classes such that each class contains exactly W/R objects, setting this limit on the size of each class to W/R seems a reasonable choice.

Consider the cases of the BAM and the Tsetlin (Krinsky) automata. In the former case the neighbours of a point X on the real line are just $X+\epsilon$ and $X-\epsilon$, for some small ϵ . In the latter case the neighbours of an action α_i are just $\alpha_{(i-1) \bmod R}$ and $\alpha_{(i+1) \bmod R}$, for $1 \leq i \leq R$. However, in the case of the OMA, we aimed to generalize the neighbours of an action α_i to be α_j , for all $1 \leq j \leq R$ and $j \neq i$. Thus, an object can move from α_i to α_j , without going through actions $\alpha_{i+1} \dots \alpha_{j-1}$. This is an additional feature of the OMA.

4.4.3 Environment

The "query system" plays the role of the environment interacting with the OMA. It rewards and penalizes the objects moving around in the OMA according to the following rules :

- (1) If the pair (A_i, A_j) is accessed, and if O_i and O_j both choose the same action, say α_k , then a reward (i.e., $\beta = 0$), will be given.
- (2) If O_i is using α_i and O_j is using α_j such that $\alpha_i \neq \alpha_j$, then a penalty (i.e., $\beta = 1$), will be given.

In the following sections, we shall discuss how this method was used to tackle the EPP.

4.4.4 Structure of the OMA

Suppose we are given W objects and intend to partition them into R classes such that each class contains W/R objects. Just like the BAM, a set of W abstract objects is associated with the physical objects and only the abstract objects are manipulated. As opposed to the Tsetlin and the Krinsky automata

in which the state of automaton can move from one action to another, in this case we have all the abstract objects **moving around** in the automaton. Thus, an OMA can be defined as a quintuple $(\{\alpha_1, \dots, \alpha_R\}, \{\phi_1, \dots, \phi_{RN}\}, \{0,1\}, L(\cdot, \cdot), G(\cdot))$, where

- (a) $\{\alpha_1, \dots, \alpha_R\}$ is the set of actions that the objects must choose from. Each action α_i represents a certain class, $1 \leq i \leq R$.
- (b) $\{\phi_1, \dots, \phi_{RN}\}$ is the set of the states. In this case, each action α_i has N states $\phi_{(i-1)N+1} \dots \phi_{iN}$ associated with it, $1 \leq i \leq R$. With no loss of generality, we can assume $\phi_{(i-1)N+1}$ to be the most internal state and ϕ_{iN} the boundary state of α_i , respectively.
- (c) $\{0, 1\}$ is the set of inputs from the environment which we described in Section 4.4.3. The value '0' represents a reward and the value '1' represents a penalty.
- (d) $L(\cdot, \cdot)$ is the transition matrix described in detail in Section 4.4.5.
- (e) The output map $G(\cdot)$ is defined as follows : If an abstract object O_i is in state ϕ_j , then the corresponding action that it chooses is

$$\alpha = \lceil \phi_j / N \rceil$$

for all $1 \leq i \leq R$ and $1 \leq j \leq RN$.

4.4.5 Method

Initially, the set of W objects are assigned arbitrarily to the R actions such that each action has exactly W/R objects assigned to it. When a pair of physical objects (A_i, A_j) is accessed, if the corresponding abstract objects O_i and O_j are choose the same action α_K , then both of them will be rewarded by moving toward the most internal state $\phi_{(K-1)N+1}$ of α_K , one step at a time. See Figure 4.3 (a). On the other hand, if they are found in different classes, say α_K and α_m

respectively (i.e., O_i is in state $\xi_i \in \{\phi_{(K-1)N+1}, \dots, \phi_{KN}\}$ and O_j is in state $\xi_j \in \{\phi_{(m-1)N+1}, \dots, \phi_{mN}\}$), then they will be penalized by moving them away from $\phi_{(K-1)N+1}$ and $\phi_{(m-1)N+1}$ as follows:

- (a) If $\xi_i \neq \phi_{KN}$ and $\xi_j \neq \phi_{mN}$, then move O_i and O_j one state toward ϕ_{KN} and ϕ_{mN} , respectively. See Figure 4.3 (b).
- (b) If exactly one is in the boundary state, (i.e., either $\xi_i = \phi_{KN}$ or $\xi_j = \phi_{mN}$, but not both), then move the object which is **not** in the boundary state toward its boundary state. See Figure 4.3 (c).
- (c) If both are in the boundary state, (i.e., $\xi_i = \phi_{KN}$ and $\xi_j = \phi_{mN}$), then one of them, say O_i , will be moved to the boundary state of α_m . In this case, since this will result in an excess of objects in α_m , one of the objects in α_m which is **not** accessed is moved to the boundary state of α_K . We choose to move the one closest to ξ_j . See Figure 4.3(d).

As in the case of the BAM, periodically (i.e., after T accesses), the physical objects are partitioned as specified by the location of the **abstract** objects. However, no clustering based on the proximity of these objects is required as in the BAM because the OMA itself specifies whether the objects are to be in the same class or not.

For the sake of completeness, we now algorithmically give the above described technique. For simplicity, as in the case of the Tsetlin and the Krinsky automata, we merely update the indices of the states in which the objects are, and the corresponding actions can be computed easily.

Figure 4.3 (a)

Figure 4.3 (b)

Figure 4.3 (c)

Figure 4.3 (d)

Figure 4.3 The Object Migrating Automaton

Algorithm OMA

Input : The abstract objects $\{O_1, \dots, O_W\}$ and the number of states N per action. Also coming into the algorithm is a stream of queries (A_i, A_j) .

Output : A periodic clustering of the objects into R partitions.

Notation : ξ_i is the state of the abstract object O_i . It is an integer in the range $1 \dots RN$, where, if $(k-1)N+1 \leq \xi_i \leq kN$, then object O_i is assigned to α_k .

Method : On the next page.

Method : Initialize ξ_i arbitrarily for $1 \leq i \leq W$, each class having W/R objects.

Repeat

For a sequence of T queries **do**

ReadQuery (A_i, A_j)

If $((\xi_i \div N) = (\xi_j \div N))$ **Then** (*The partitioning is rewarded*)

If $((\xi_i \bmod N) \neq 1)$ **Then** (*Move O_i towards the internal state*)
 $\xi_i = \xi_i - 1$

EndIf

If $((\xi_j \bmod N) \neq 1)$ **Then** (*Move O_j towards the internal state*)
 $\xi_j = \xi_j - 1$

EndIf

Else (* The partitioning is penalized *)

If $((\xi_i \bmod N) \neq 0)$ **and** $((\xi_j \bmod N) \neq 0)$ **Then**
 (* Both are in internal states *)

$\xi_i = \xi_i + 1$

$\xi_j = \xi_j + 1$

Else If $(\xi_i \bmod N \neq 0)$ **Then** (* O_i is in an internal state*)
 $\xi_i = \xi_i + 1$

Else If $(\xi_j \bmod N \neq 0)$ **Then** (* O_j is in an internal state*)
 $\xi_j = \xi_j + 1$

Else (* Both are in boundary states *)

temp $\leftarrow \xi_i$ (* Store the state of O_i *)

$\xi_i = \xi_j$ (* Move O_i to same group as O_j *)

$t =$ index of an **unaccessed** object in group of O_j
 where O_t is closest to ξ_j

$\xi_i = \text{temp}$ (* Move O_i to the old state of O_i *)

EndIf

EndIf

EndFor

Print out partitions based on the states ξ_i

ForEver

End Algorithm OMA

4.4.6 An OMA Example

Consider the same example of partitioning a set of four objects $\{A_1, A_2, A_3, A_4\}$ into two equal-size classes as described in Section 4.3.5. Since $W = 4$ and $R = 2$, the number of objects per class is $W/R = 2$. Thus a two-action OMA is needed. In this case, each action has two states associated with it. For simplicity, in the illustrations below, the indices of the objects represent the abstract objects.

Formally, the set of actions is defined as follows :

α_1 : has two states, namely ϕ_1 and ϕ_2

α_2 : has two states, namely ϕ_3 and ϕ_4

The queries are :

Query 1 : Access objects A_1 and A_2 .

Query 2 : Access objects A_1 and A_3 .

Query 3 : Access objects A_1 and A_2 .

Query 4 : Access objects A_3 and A_4 .

Query 5 : Access objects A_1 and A_2 .

Step 0 : The four abstract objects are arbitrarily assigned to the two actions α_1 and α_2 such that O_1, O_4 are in α_1 and O_2, O_3 are in α_2 .

Step 1 : The pair (A_1, A_2) is accessed. Since O_1 and O_2 are in α_1 and α_2 respectively, both of them are penalized. Besides, since O_2 is in the boundary state and O_1 is not, only O_1 is moved toward the boundary state of α_1 .

Step 2 : The pair (A_1, A_3) is accessed. Since O_1 and O_3 are in α_1 and α_2 respectively, both of them are penalized. Besides, since both of them are in boundary states, only one of them will be moved. Suppose O_3 is the one to be moved to α_1 . Then one of the objects in α_1 not being accessed will be moved to α_2 . In this case, O_4 is the only choice and is moved from α_1 to α_2 .

Step 3 : The pair (A_1, A_2) is accessed. This situation is the same as that in step 2 (i.e., both are penalized). This time, O_2 decides to move to α_1 , and O_3 decides to leave α_1 and enter α_2 .

Step 4 : The pair (A_3, A_4) is accessed. Since O_3 and O_4 are both in α_2 , they are rewarded by moving toward the most internal state of α_2 (i.e., ϕ_3).

Step 5 : The pair (A_1, A_2) is accessed. Just like step 4, O_1 and O_2 are rewarded and moved toward state ϕ_1 , the most internal state of α_1 .

Step 6 : The algorithm can be considered to have converged since all the objects are in the most internal states of the actions chosen. Hence, the partition $\{(A_1, A_2), (A_3, A_4)\}$ is regarded as the solution partition.

4.4.7 Analytical Results Concerning the OMA

A rigorous analysis of the OMA in terms of its convergence accuracy is by no means trivial. In its truest sense, the OMA is a strategy involving an interaction of the query system **and** the various objects, and thus in this perspective, it must be viewed as a co-operative automaton game in which the objects try finding the optimal partitioning among themselves. Although a rigorous analysis of the OMA is not available, based on the experimental results we have obtained, we conjecture that the scheme is ϵ -optimal.

In the special case when $W = 4$ and $R = 2$, we shall now show that the OMA is expedient, even when N is as small as unity.

Theorem 4.3

When $W = 4$ and $R = 2$, the Object Migrating Automaton with $N = 1$ is expedient.

Proof :

Let the objects be A_1, A_2, A_3, A_4 , and let the probability of a query (A_i, A_j) be given as $\Pr[(A_i, A_j)]$. We define the following quantities :

$$C_{12} = \Pr[(A_1, A_2)] + \Pr[(A_3, A_4)]$$

$$C_{13} = \Pr[(A_1, A_3)] + \Pr[(A_2, A_4)]$$

$$C_{14} = \Pr[(A_1, A_4)] + \Pr[(A_2, A_3)]$$

Note that $\sum_{2 \leq i \leq 4} C_{1i} = 1$. For the problem to be meaningful, there must be a partitioning superior to the others. Hence, one of the C_{1i} 's must be strictly greater than $1/3$, and the sum of the other two strictly less than $2/3$.

The Markov chain that describes the OMA is thus given by three states $\{\omega_{12}, \omega_{13}, \omega_{14}\}$ where ω_{1i} represents the condition that A_1 should be grouped together with A_i . Notice that if i, j and k are distinct indices in the set $\{2, 3, 4\}$, then both the queries (A_1, A_i) and (A_j, A_k) support the partitioning represented by ω_{1i} . Thus the transition matrix of the Markov chain of the OMA is given by the matrix F^* , where,

$$F^* = \begin{bmatrix} 1-C_{13}-C_{14} & C_{13} & C_{14} \\ C_{12} & 1-C_{12}-C_{14} & C_{14} \\ C_{12} & C_{13} & 1-C_{12}-C_{13} \end{bmatrix}$$

The chain is ergodic, and hence, using the theory of Markov chains, the limiting state occupation probabilities obey the equation,

$$\Pi = (F^*)^T \Pi \quad (4.7)$$

where $\Pi = [\Pi_2, \Pi_3, \Pi_4]^T$, and $\Pi_i = \Pr[\text{state as } t \rightarrow \infty \text{ is } \omega_{1i}]$

The system of equations represented by (4.7) are :

$$\begin{aligned} (-C_{13} - C_{14})\Pi_2 + C_{12}\Pi_3 + C_{12}\Pi_4 &= 0 \\ C_{13}\Pi_2 + (-C_{12} - C_{14})\Pi_3 + C_{13}\Pi_4 &= 0 \\ C_{14}\Pi_2 + C_{14}\Pi_3 + (-C_{12} - C_{13})\Pi_4 &= 0 \end{aligned}$$

Solving the above sets of equations and simplifying yields :

$$\Pi_i = C_{1i} / (C_{12} + C_{13} + C_{14}) \quad (4.8)$$

Observe that the OMA is expedient if the limiting loss is strictly less than the initial loss. Using (4.8), we write the expression for the limiting loss, $M(\infty)$, as :

$$M(\infty) = \frac{(C_{13} + C_{14})C_{12} + (C_{12} + C_{14})C_{13} + (C_{12} + C_{13})C_{14}}{C_{12} + C_{13} + C_{14}} \quad (4.9)$$

Similarly, if each action is initially chosen with probability 1/3, we write,

$$M(0) = 1/3 [(C_{13} + C_{14}) + (C_{12} + C_{14}) + (C_{12} + C_{13})] \quad (4.10)$$

Comparing (4.9) and (4.10), we aim to prove that

$$\frac{2(C_{12}C_{13} + C_{12}C_{14} + C_{13}C_{14})}{C_{12} + C_{13} + C_{14}} < \frac{2(C_{12} + C_{13} + C_{14})}{3}$$

or in other words, that

$$3(C_{12}C_{13} + C_{12}C_{14} + C_{13}C_{14}) < (C_{12} + C_{13} + C_{14})^2 \quad (4.11)$$

But $C_{12} + C_{13} + C_{14} = 1$. Thus, the OMA is expedient if

$$C_{12}C_{13} + C_{13}C_{14} + C_{12}C_{14} < 1/3. \quad (4.12)$$

But, by Lemma 4.1 and the fact that all the C_{1i} 's cannot be equal to 1/3, (4.12) is true. Hence the result. ♦ ♦ ♦

Lemma 4.1

Let $p, q, r \geq 0$, such that $p+q+r = 1$. Then, $pq + qr + rp$ has a maximum value of $1/3$, this value being achieved when $p = q = r = 1/3$.

Proof :

$$\begin{aligned} \text{Let } f(p,q) &= pq + qr + rp \\ &= pq + (p + q) (1 - (p+q)) \end{aligned}$$

We maximize $f(\cdot)$ by taking partial derivatives with respect to p and q . Thus,

$$\partial f(\cdot) / \partial p = 0 \Rightarrow 2p + q = 1 \quad (4.13)$$

$$\text{Similarly, } \partial f(\cdot) / \partial q = 0 \Rightarrow 2q + p = 1 \quad (4.14)$$

Solving (4.13) and (4.14) yields the result that $f(\cdot)$ is maximized when $p = q = r = 1/3$. Further the maximum value of $f(\cdot)$ is exactly $1/3$. Hence the lemma.

◆◆◆

From the above, we observe that for larger values of W and R , the proof concerning the ϵ -optimality of the scheme will definitely be non-trivial.

4.4.8 Experiments

The BAM and the OMA were compared experimentally with the objectives to:

- (a) compare the rate of convergence and accuracy of the two methods for various numbers of objects (W) and class sizes (R), and to
- (b) compare the rate of convergence and accuracy of the two methods for different joint access probabilities.

In case (a), one hundred experiments were conducted for each combination of

W and R. The joint access probability was 0.9. In case (b), the values of W and R were set to six and three, respectively, and the joint access probability P, as defined by (2.5), was in the range 0.6 .. 0.9 with a step of size 0.05. Again, one hundred experiments were performed for each value of P.

4.4.8.1 Initialization

Initially, each of the W given objects was assigned arbitrarily to an action α_i , $1 \leq i \leq K$, such that each action has exactly W/R objects. The initial set up of the BAM was the same as that described in Section 2.2.7.1.

4.4.8.2 Parameters and Assumptions

There are two parameters involved in the OMA, namely the number of states per action and the class size. In all the experiments performed, these two parameters were set up as follows :

- (a) The number of states per action was set to 10.
- (b) If W objects were given and were supposed to be partitioned into R classes, then the class size was set to W/R.

The parameter set up for the BAM was the same as that described in Section 2.2.7.2, and the assumptions used were the same as that described in Section 2.2.7.3.

4.4.8.3 Results

A summary of the results obtained is found in Tables 4.2 and 4.3. From Table 4.2, we observe the superiority of the OMA to the BAM. For example, when $W = 18$ and $R = 3$, on the average, the BAM converged to the optimal solution after 12756 iterations. The OMA recorded the corresponding figure as

5783. Notice that this implies an improvement by a factor of almost 2.2. Further, the OMA provided correct partitioning in all the experiments performed. But in case of the BAM, as mentioned in Section 2.2.6, when $W = 12$ and $R = 2$, among the one hundred experiments performed, only eighty-one gave the expected partitioning and the other nineteen did not. In the cases when the experiments failed, one of the expected classes was actually partitioned into two sub-classes. A similar situation was also reported in [27]. A comparative graphical sketch of the average number of iterations required for convergence of the BAM and the OMA is shown in Figure 4.4 as a function of the number of objects. In this case, for each point on the graph the number of classes is three. The advantage of the OMA is obvious.

From Table 4.3, we observe that both the OMA and the BAM possessed excellent accuracy (i.e., 100 per cent), in all the experiments performed, for various joint access probabilities P defined by (2.5). In general, the OMA again converged faster than the BAM, but the difference in the rate of convergence decreased as the joint access probability decreased.

No. Objects	Objects/Class	No. Classes	# iterations BAM	# iterations OMA
4	2	2	(21, 40)	(18, 45)
6	2	3	(255, 269)	(33, 73)
	3	2	(243, 251)	(115, 160)
9	3	3	(745, 762)	(209, 294)
12	2	6	(1520, 1527)	(70, 163)
	3	4	(1885, 1894)	(243, 350)
	4	3	(2914, 2925)	(569, 677)
	6	2	(3593, 3609)	(1993, 2092)
15	3	5	(2501, 2508)	(317, 454)
	5	3	(6442, 6445)	(2000, 2192)
18	2	9	(3348, 3352)	(111, 277)
	3	6	(2910, 2917)	(334, 499)
	6	3	(12754, 12756)	(5169, 5783)
	9	2	Not Available	(10096, 10268)

Table 4.2 Comparison of the BAM and the OMA Solutions of the EPP.

Notation : Number of iterations is given as a pair (x,y), where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning.

Figure 4.4 Plot of average number of iterations required for convergence for the BAM and the OMA as the number of objects increases. In each case number of classes is three.

P	# iterations BAM	# iterations OMA	Accuracy BAM	Accuracy OMA
0.9	(255, 269)	(33, 73)	100%	100%
0.85	(272, 285)	(34, 85)	100%	100%
0.8	(322, 332)	(35, 99)	100%	100%
0.75	(341, 349)	(38, 122)	100%	100%
0.7	(375, 385)	(39, 153)	100%	100%
0.65	(414, 421)	(44, 225)	100%	100%
0.6	(493, 501)	(63, 513)	100 %	100%

Table 4.3 Comparison of the BAM and the OMA solutions for various joint access probabilities. In each case, the number of objects is six and the number of classes is three.

Notation : Same as Table 4.2

4.4.9 Complexity of the OMA

If the OMA processes T_{OMA} queries to compute its solutions, then the time complexity of the OMA algorithm will be $O(T_{OMA})$. Since the OMA algorithm needs W memory locations to store the states of the W abstract objects and R locations to store the size of the R classes at any time instant, the space complexity of the OMA is $O(S_{OMA})$ where $S_{OMA} = W + R$. Clearly, these complexities are practical and feasible.

Based on the experimental results we have obtained, we believe that, in general

$$O(T_{OMA}) \ll O(T_{BAM}) \quad \text{and}$$

$$O(S_{OMA}) = O(S_{BAM}).$$

4.5 Concluding Remarks

In this chapter we have used three deterministic automata solutions, namely the Tsetlin, the Krinsky and the Object Migrating Automata, to handle a special case of the OPP which we have called the Equal Partitioning Problem (EPP). Although the first two automata are ϵ -optimal, they are feasible only when the size of the object set is small. On the other hand, the OMA is feasible even when the object size is large. The OMA is, in general, faster than the BAM.

In the next chapter, we shall proceed to study the OPP by first generalizing the three solutions that we have discussed in this chapter. Following that another non-deterministic FSSA will be discussed.

Chapter 5

FSSA Approaches To The OPP

Having studied the Equal Partitioning Problem (EPP) in last chapter, we would like to proceed to the discussion of the Object Partitioning Problem (OPP). Observe that the objective of the OPP is to partition a set of W objects into R classes such that objects that are accessed more frequently together lie in the same class. In this case each class may have a **different** number of objects because we assume that the underlying unknown grouping specified by the user's queries is not based on equal-size groups.

Since the OPP is the generalization of the EPP, we would like to first investigate the possibility of generalizing the three automata solutions to the EPP proposed earlier. Specifically, we would first like to investigate whether the Tsetlin, the Krinsky and the Object Migrating Automata (OMA), can handle the OPP before searching for completely new automata solutions.

5.1 Generalization of the Tsetlin and Krinsky Automata Solutions

The Tsetlin and the Krinsky automata discussed in Chapter Four solved the EPP by representing each of the possible equal-size partitions that can be obtained from the given set of objects by an action offered by the interacting environment. The automaton then endeavours to find the optimal action representing the optimal partitioning of the objects.

5.1.1 Method

Since the difference between the EPP and the OPP is that the former requires that each class has the same number of objects while the latter does

2 OF/DE 2

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

not have this requirement, the same approach that has been used to solve the EPP can be used to solve the OPP. The method is as follows :

Each of the possible partitions, both of **equal-** and **unequal-size**, that can be obtained from the given set of objects is represented by an action α_i offered by the interacting environment. Then by processing the queries (i.e., interacting with the environment), the automaton is used to learn the optimal action, which in this case is the optimal partitioning of the objects.

5.1.2 Problems with the Tsetlin and the Krinsky Approach

Although this method is simple, as discussed in Theorem 4.3, partitioning a set of W objects into R equal-size classes requires that a Tsetlin or a Krinsky automaton possesses $W! / [(M!)^R R!]$ actions, where $M = W/R$. If unequal-size partitions are also taken into consideration, then the automaton is required to possess K actions, where K is much larger than $W! / [(M!)^R R!]$. Unless there is a straightforward way to code the actions into their partition representations such that on processing a query, a linear search will not have to be done, these partition representations must be stored in a static memory array. Since the number of partitions grows exponentially as the size of the object set increases, clearly, using the Tsetlin or the Krinsky automaton to handle the OPP is feasible **only** when the number of objects is small.

5.2 Generalization of the OMA Approach

As opposed to the Tsetlin and the Krinsky automata models, generalizing the OMA to handle the OPP is certainly non-trivial because the heart of concepts justifying the OMA is founded on the limit imposed on "class size". This limit is used to prevent two or more groups of objects from merging into

one single class. In the case of the ~~EPP~~, this limit can be set easily to W/R if W objects are to be partitioned into R classes. In the case of the OPP, however, the value W/R can no longer be used since each class may or may not have exactly W/R objects in it.

5.2.1 Method

Due to the fact that each class may have a different number of objects, one may come up with the idea of setting an **upper bound s** on the class size instead of a fixed limit on it (i.e., at each time instant, each class can have **at most s** objects). Just like the OMA discussed before, if an object O_i leaves action α_i and enters action α_j which already has s objects, then one of the objects in α_j not being accessed will be moved to α_i .

5.2.2 Problem

Consider the following elementary question: can the use of an upper bound on the class size prevent different groups of objects from entering the same class? If the answer of the question were in the affirmative, this would lead to a possible solution of the OPP. However, this is not the case because of the following theorem.

Theorem 5.1

In the OMA, setting an upper bound on the class size does not **always** prevent more than one group of objects from entering the same class.

Proof :

Suppose W objects are given and these objects are partitioned into R classes, not necessarily of equal size, based on the queries against them. The query statistics are, of course, unknown to the OMA. Consider the query system in which the queries support the partitioning where the group $\{O_1, \dots, O_m\}$ is in one class, and $\{O_{m+1}, \dots, O_{m+n}\}$ is in another class. We denote the former group as G_1 , of size m (i.e., $|G_1| = m$), and the latter group as G_2 , of size n (i.e., $|G_2| = n$), where $m, n > 0$. With no loss of generality, $m, n \geq 2$. Further, suppose the queries also support the group $G_3 = \{O_{m+n+1}, \dots, O_{2m+2n}\}$, of size $m+n$. Since no *a priori* information is given, we assume that every class has the same upper bound S , where $S \geq m+n$.

Suppose at the k^{th} time instant $O_i, O_j \in G_1$ are in classes (represented by actions) α_p and α_q , respectively. Besides, all the n objects of G_2 are also in α_q (i.e., α_q has $n+1$ objects). Consider the situation when O_i and O_j are in the boundary states of α_p and α_q , respectively. If the next query requests the pair (A_i, A_j) , then according to the transition map of the OMA, either O_i or O_j will move to the action where the other object is in. With no loss of generality, O_i is the one to leave α_p and enter α_q . Adding O_i to α_q makes α_q contain $n+2$ objects. Since $n+2 < S$ (because $2 \leq m$ and $m+n < S$) and α_q can have at most S objects, no objects in α_q is required to move to α_p .

In this case, if $m = 2$, then we have both G_1 and G_2 found in the same action, namely α_q . Since this implies that the elements of G_1 and G_2 are in the same class, they will be rewarded quite frequently, and will eventually reach the most internal state of α_q leading to an erroneous partition. Hence the theorem is proved. ♦ ♦ ♦

Thus, setting an upper bound on the class size in the OMA does not seem helpful in solving the OPP.

Since the Tsetlin, the Krinsky and the OMA solutions do not lead to feasible solutions to the OPP, searching for new automata is necessary. In the following sections, we shall propose another FSSA which we have called the Stochastic Move Automaton (SMA). Although it turns out that this method cannot handle the OPP properly, it is certainly a stepping stone to the development of the automaton that we shall discuss in the next chapter.

5.3 Two Important Properties

Before proceeding to the discussion of the SMA, we would like to discuss two important properties which, in our opinion, are essential for finding automata solutions to the OPP.

Property 5.1

Initially assigning more than one object to a class **could** result in having more than one group of objects in the same class.

Reason :

Suppose a set of W abstract objects $\{O_1, \dots, O_W\}$ is given and R classes (represented by actions) are available for them with $R \leq W$. Since $R < W$ and each object has to be assigned to one class, obviously, there will be at least one class with more than one object. Consider the case when O_i and O_j are in action α_k and the queries access the pairs (A_i, A_m) and (A_j, A_n) more frequently than the pairs (A_i, A_j) , (A_i, A_n) and (A_j, A_m) . In this case, if the abstract objects O_m and O_n eventually entered action α_k , they could stay with O_i and O_j . This

implies that O_i, O_j, O_m and O_n are in the same class, namely α_k . However, the fact that the pairs $(A_i, A_j), (A_i, A_n)$ and (A_j, A_m) are seldom accessed and the pairs (A_i, A_m) and (A_j, A_n) are frequently accessed indicates that O_i and O_m should be in one class, say α_p , and O_j and O_n should be in another class α_q , where $\alpha_p \neq \alpha_q$. However, by the merging of the abstract objects O_i, O_j, O_m and O_n into the same class α_k , such a subpartitioning is impossible and the property follows.

Property 5.2

One **improper** movement of an object in an automaton can result in having more than one group of objects found in the same class.

Reason :

Consider the case when a set of abstract objects $\{O_1, \dots, O_W\}$ is given and they are initially assigned to choose different classes (i.e., actions, each action has only one object assigned to it). Let objects O_i, O_j and O_k be in classes α_p, α_q , and α_r , respectively, where $1 \leq p, q, r \leq W$ and $p \neq q \neq r$. Suppose the queries access the pair (A_i, A_j) more frequently than the pairs (A_i, A_k) and (A_j, A_k) . Then objects O_i and O_j should lie in the same class and O_k should lie in another class. In order to achieve this goal, either O_i or O_j or both are required to change their classes, simply because they are in different classes. Without loss of generality, we can assume that O_i is the object about to change its class (i.e., from α_p to α_q). However, as discussed in Section 2.2.7.3, the P value defined by (2.5) can be less than one. This implies that queries may access objects that are in different groups. If the pair (A_i, A_k) is accessed instead of the pair (A_i, A_j) , then O_i which is about to leave from α_p will enter α_r instead of α_q .

This can also happen to O_j and will result in having O_i and O_j in α_r , which is the class for another group of objects. Hence the theorem. ♦ ♦ ♦

Although it appears as if these two criteria concern two different aspects of the problem, in actuality they both play important roles in finding solutions to the OPP because one affects the performance of the other. We shall now proceed to the discussion of the SMA which uses these two properties in its structure.

5.4 The Stochastic Move Automaton

5.4.1 Approach

In order to avoid having the initialization problem discussed in Theorem 5.2, the following method is proposed: Instead of assigning W objects to R classes, $R < W$, such that each class may have more than one object in it, the W objects are initially assigned to W **distinct** classes so that no two objects are found in the same class. Since all the objects are in different classes, if there is no "incorrect" movement during the process of partitioning objects, then obviously no two groups of objects will be found in the same class.

On the other hand, moving objects around in an automaton is not trivial due to the fact that the P value defined by (2.5) can be less than 1. This in turn implies that objects that actually should be in different groups may be accessed together. Observe that such queries can be regarded as "incorrect" because they provide misleading information for objects that are moving around in the automaton. Although the existence of such queries is certain, it is **impossible** to detect them in advance, simply because the ideal grouping of the objects is unknown *a priori*. Hence, the best we can do is to hope to "live with them".

Suppose we come to a partitioning for which no object is required to change its action. Clearly, in this case, an "incorrect" query will not affect this partitioning. However, if we do not come to such a partitioning, according to Property 5.2, such a query may have adverse effects on the outcome, namely it can lead to an incorrect partitioning of the objects. To avoid this, one reasonable way is to try to ignore such a query. This is equivalent to giving the abstract objects the option of either following the information given by the query or ignoring the information. Thus, in general, an object O_i changes classes, say from α_p to α_q , with probability δ and stays in α_p with probability $1 - \delta$. Specifically,

$$\Pr [O_i \text{ moves out from } \alpha_p] = \delta \quad (5.1)$$

$$\Pr [O_i \text{ stays in } \alpha_p] = 1 - \delta \quad (5.2)$$

We shall now discuss the SMA which employs the two strategies mentioned above to handle the OPP.

5.4.2 Structure of the SMA

Let $\{A_1, \dots, A_W\}$ be the set of W given objects. Just like the OMA, the set of abstract objects are moving around in the SMA. An SMA can be defined as a quintuple $(\alpha, \Phi, \beta, F, G)$, where

- (i) $\alpha = \{\alpha_1, \dots, \alpha_W\}$ is the set of actions that the abstract objects must choose from. Each action α_i , $1 \leq i \leq W$, represents a certain class.
- (ii) $\Phi = \{\phi_1, \dots, \phi_{WN}\}$ is the set of WN states. Each action α_p has N states associated with it. With no loss of generality, we can assume $\phi_{(p-1)N+1}$ is the most internal state and ϕ_{pN} is the boundary state of α_p .
- (iii) $\beta = \{0, 1\}$ is the set of inputs from the environment. The value '0'

represents a reward and the value '1' represents a penalty.

- (iv) Let ξ_i and ξ_j be the states of the abstract objects O_i and O_j , respectively, and let the query be (A_i, A_j) . Then the transition matrix F is defined as follows:

$$\beta = 0 \quad \begin{cases} \xi_k = \xi_k - 1 & \text{if } (\xi_k \bmod N) \neq 1 & \text{for } k = i, j & (5.3) \\ \xi_k = \xi_k & \text{if } (\xi_k \bmod N) = 1 & \text{for } k = i, j & (5.4) \end{cases}$$

$$\beta = 1 \quad \begin{cases} \xi_k = \xi_k + 1 & \text{if } (\xi_k \bmod N) \neq 0 & \text{for } k = i, j & (5.5) \end{cases}$$

$$\xi_k = \lceil \xi_m / N \rceil * N \quad \text{if } (\xi_k \bmod N) = 0 \quad \text{for } k = i, j, m = i, j, k \neq m \quad (5.6)$$

The transitions (5.3), (5.4) and (5.5) are with probability 1.0 while the transition (5.6) is with probability δ .

- (v) If O_i is in state ξ_i and α_p is the action that O_i is in, then G , the output matrix, is defined by the following equation:

$$p = \lceil \xi_i / N \rceil \quad \text{for } 1 \leq i \leq W$$

5.4.3 Environment

Just like the OMA, the "query system" plays the role of the environment with which the SMA interacts. At each time instant, if (A_i, A_j) is the pair of physical objects accessed and the corresponding abstracts objects O_i and O_j are in action α_p and α_q , respectively, then the environment will send out

- (a) a reward (i.e., $\beta = 0$), if $\alpha_p = \alpha_q$
- (b) a penalty (i.e., $\beta = 1$), if $\alpha_p \neq \alpha_q$

5.4.4 Method

Initially, each of the W abstract objects O_i is assigned to an action α_k , $1 \leq i$, $k \leq W$. Without loss of generality, we can assume $i = k$. At each time instant, a

pair of physical objects (A_i, A_j) is accessed, and if the corresponding abstract objects O_i and O_j are in the same action, say α_k , they will be rewarded and will move toward the most internal state of α_k . On the other hand, if O_i and O_j are in different actions, say α_p and α_q , then they will be penalized and will move toward the boundary states of α_p and α_q , respectively. If either O_i or O_j or both need to change action, then the moving object(s) will move to another action with a probability δ . Just like the OMA, if O_i intends to leave α_p and enter α_q , it can just move to α_q without going through the intermediate actions $\alpha_{(p+1) \bmod W}, \dots, \alpha_{(q-1) \bmod W}$.

After T accesses, the physical objects are reorganized, if necessary, based on the actions chosen by the corresponding abstract objects. For the sake of completeness, we present below an algorithmic version of the SMA. As before, we only keep track of the states of the objects.

Algorithm SMA

Input : The abstract objects $\{O_1, \dots, O_W\}$, the number of states N per action, and δ , the probability of changing actions. Also coming into the algorithm is a stream of queries (A_i, A_j) .

Output : A periodic partitioning of the objects.

Notation : ξ_i is the state of the abstract object O_i . It is an integer in the range $1, \dots, WN$, where, if $(k-1)N + 1 \leq \xi_i \leq kN$, then object O_i is assigned to α_k . The motion of O_i and O_j at the boundary state is determined by invoking a random number generator $RAND$.

Method : Initialize ξ_i for $1 \leq i \leq W$ such that each class has only one object.

Repeat

For a sequence of T queries **do**

ReadQuery (A_i, A_j)

If $((\xi_i \text{ div } N) = (\xi_j \text{ div } N))$ **Then** (*The partitioning is rewarded*)

If $((\xi_i \text{ mod } N) \neq 1)$ **Then** (*Move O_i towards the internal state*)
 $\xi_i = \xi_i - 1$

EndIf

If $((\xi_j \text{ mod } N) \neq 1)$ **Then** (*Move O_j towards the internal state*)
 $\xi_j = \xi_j - 1$

EndIf

Else (* The partitioning is penalized *)

If $((\xi_i \text{ mod } N) \neq 0)$ and $((\xi_j \text{ mod } N) \neq 0)$ **Then**

(* Both are in internal states *)

$\xi_i = \xi_i + 1$

$\xi_j = \xi_j + 1$

Else If $(\xi_i \text{ mod } N \neq 0 \text{ AND } \xi_j \text{ mod } N = 0)$ **Then**

$\xi_i = \xi_i + 1$ (* O_i is in an internal state *)

If $(\text{RAND} < \delta)$ **Then** $\xi_j = \lceil \xi_j / N \rceil * N$

Else If $(\xi_i \text{ mod } N = 0 \text{ AND } \xi_j \text{ mod } N \neq 0)$ **Then**

$\xi_j = \xi_j + 1$ (* O_j is in an internal state *)

If $(\text{RAND} < \delta)$ **Then** $\xi_i = \lceil \xi_i / N \rceil * N$

Else (* Both are in boundary states *)

temp1 = ξ_i

temp2 = ξ_j

If $(\text{RAND} < \delta)$ **Then** $\xi_i = \text{temp2}$

If $(\text{RAND} < \delta)$ **Then** $\xi_j = \text{temp1}$

EndIf

EndIf

EndFor

Print out partitions based on the states ξ_i

Forever

End Algorithm SMA

5.4.5 An SMA Example

Consider an example of partitioning a set of four objects $\{A_1, A_2, A_3, A_4\}$ into two classes. If the queries against these objects request the pairs (A_1, A_2) and (A_3, A_4) more frequently than any other pair, then it is desirable to have A_1, A_2 in one class and A_3, A_4 in another class. Since four objects are given, a four-action SMA is required. For simplicity, each action has only one state associated with it. In the illustrations, we shall only use indices to represent objects (i.e., 'i' for O_i).

The set of queries is :

Query 1 : Access objects (A_1, A_2) .

Query 2 : Access objects (A_1, A_4) .

Query 3 : Access objects (A_3, A_4) .

Step 0 : Initially, each object O_i is assigned to action α_i , for $1 \leq i \leq 4$.

Step 1 : The pair (A_1, A_2) is accessed. Since O_1 and O_2 are in actions α_1 and α_2 , respectively, they are both penalized. Besides, since they are in the boundary states of the using actions, O_1 can move to α_2 and O_2 can move to α_1 . Suppose O_2 decides to move and O_1 decides to stay, then O_2 will enter α_1 and stay together with O_1 .

Step 2 : The pair (A_1, A_4) is accessed. Again, since O_1 and O_4 are in different actions (i.e., α_1 and α_4), they are both penalized. If O_4 decides to move to α_1 and O_1 decides to stay in α_1 , then O_1 , O_2 and O_4 will be found in α_1 by the end of this step.

Step 3 : The pair (A_3, A_4) is accessed. O_3 and O_4 are penalized since they are in different actions (i.e., α_3 and α_4). If O_4 decides to move to α_3 and O_3 decides to stay in α_3 , then after this step both O_3 and O_4 will be found in the same action, namely α_3 .

Step 4 : The algorithm can be regarded as having converged because O_1 and O_2 are in the most internal state of α_1 and O_3 and O_4 are in the most internal state of α_3 . Of course, this is just a trivial example in which $N = 1$. In general, the transitions are more varied, and the convergence is not so rapid.

5.4.6 Experiments

The SMA discussed above was simulated and compared with the BAM. The objectives were primarily to :

- (a) compare the rate of convergence and the accuracy of both methods as a function of the object size when objects are partitioned into classes, with different values of δ defined by (5.1), and
- (b) obtain the accuracy of both methods as a function of joint access probability defined by (2.5), with different values of δ .

In case (a), for each W (the number of objects), R (the number of classes, not necessarily of equal-size) and δ , one hundred experiments were performed in order to obtain accurate results. In case (b), one hundred experiments were also performed for each set of values of P and δ .

5.4.6.1 Initialization

If W objects were given, then initially they were assigned arbitrarily to W distinct classes (actions) so that no two objects were assigned to the same class. The initialization of the BAM was the same as that described in Section 2.2.7.1.

5.4.6.2 Parameters and Assumptions

The two parameters involved in the SMA are the number of states per action and δ , the probability of changing action for each object. In all the experiments performed, these two parameters were set up as follows :

- (a) Each action had 10 states associated with it.
- (b) The value of δ was set to 0.5 and 0.25 in order to see the effect of δ on the accuracy of the SMA solution.

The parameters of the BAM were the same as that described in Section 2.2.7.2. The assumptions made were as in Section 2.2.7.3.

5.4.6.3 Convergence

Just like the OMA, the SMA was considered to have converged to the optimal partitioning if all the objects moving around in the automaton were in the most internal states of the actions chosen.

5.3.6.4 Results

A summary of the results obtained is found in Tables 5.1, 5.2 and 5.3. From Table 5.1 (a)–(b), we observe that the SMA converged much faster than the BAM. For example, when fifteen objects were partitioned into four unequal-size classes, the BAM converged after 3507 iterations. The corresponding figures for the SMA are 369 when $\delta = 0.25$ and 307 when $\delta = 0.5$. This implies an improvement by a factor of about 9.5 and 11.4 for $\delta = 0.25$ and 0.5, respectively. A comparative graphical sketch of the BAM and the SMA is drawn in Figure 5.1 as a function of the number of objects. In this case, for each point on the graph the number of classes is three.

Unfortunately however, the accuracy of the SMA is not as good as that of the BAM. As stated earlier, an experiment is said to have "failed" if it converged to a partition which is different from the underlying partitioning used in simulating the algorithm. From Table 5.2 (a)–(b), we observe that the BAM gave 100 per cent accuracy in all but two sets of experiments. In the case of the SMA, however, when $\delta = 0.5$, the accuracy ranges from 19 per cent to 88 per cent while the mean is 73.8 per cent. Further, when $\delta = 0.25$, the accuracy ranges from 33 per cent to 87 per cent while the mean is 74.76 per cent. Figure

5.2 illustrates the accuracy of the BAM and the SMA when $\delta = 0.5$ and 0.25 as a function of the number of objects. In this case, again, for each point on the graph, the number of classes is three.

The accuracy of the SMA and the BAM was also compared for various values of P when six objects were partitioned into three equal-size classes. From Table 5.3 we observe that the accuracy of the SMA decreased as P , the joint access probability defined by (2.5), decreased. For instance, when $P = 0.9$, the SMA gave 82 per cent accuracy for both $\delta = 0.25$ and 0.5 . However, when $P = 0.75$, it only gave 47 per cent and 43 per cent accuracy for $\delta = 0.25$ and $\delta = 0.5$, respectively, and decreased to a pitiful 14 per cent and 5 per cent when $P = 0.6$. On the other hand, the accuracy of the BAM was **very stable** and maintained the value of 100 per cent even when P was decreasing. Figure 5.3 illustrates a comparative graphical sketch of the accuracy of the BAM and the SMA as a function of P . In this case, for each point on the graph the number of classes is three.

No. Objects	Objects/Class	No. Classes	# iterations BAM	# iterations SMA
4	2	2	(21, 40)	(28, 54)
6	2	3	(255, 269)	(50, 85)
	2, 4	2	(55, 70)	(85, 137)
	3	2	(243, 251)	(62, 98)
9	3	3	(745, 762)	(91, 156)
	2, 3, 4	3	(365, 377)	(136, 239)
	4, 5	2	(711, 728)	(128, 202)
12	2	6	(1520, 1527)	(98, 187)
	2, 2, 2, 3, 3	5	(1940, 1977)	(154, 290)
	3	4	(1885, 1894)	(138, 231)
	2, 3, 3, 4	4	(8193, 8197)	(192, 332)
	4	3	(2914, 2925)	(165, 265)
	6	2	(3593, 3609)	(201, 290)
15	3	5	(2501, 2508)	(182, 305)
	2, 3, 3, 3, 4	5	(8224, 8281)	(259, 430)
	3, 4, 4, 4	4	(3380, 3507)	(233, 369)
	5	3	(6442, 6445)	(251, 374)
18	2	9	(3348, 3352)	(186, 368)
	3	6	(2910, 2917)	(230, 385)
	6	3	(12754, 12756)	(331, 474)
	9	2	Not Available	(406, 543)

Table 5.1 (a) Comparison of the rate of convergence of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P , the probability defined by (2.5), is 0.9 and δ , the probability defined by (5.1), is 0.25

Notation : Number of iterations is given as a pair (x, y) , where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning.

No. Objects	Objects/Class	No. Classes	# iterations BAM	# iterations SMA
4	2	2	(21, 40)	(25, 51)
6	2	3	(255, 269)	(46, 79)
	2, 4	2	(55, 70)	(70, 115)
	3	2	(243, 251)	(50, 84)
9	3	3	(745, 762)	(78, 138)
	2, 3, 4	3	(365, 377)	(105, 206)
	4, 5	2	(711, 728)	(101, 175)
12	2	6	(1520, 1527)	(87, 172)
	2, 2, 2, 3, 3	5	(1940, 1977)	(116, 247)
	3	4	(1885, 1894)	(103, 192)
	2, 3, 3, 4	4	(8193, 8197)	(139, 284)
	4	3	(2914, 2925)	(133, 230)
	6	2	(3593, 3609)	(158, 255)
15	3	5	(2501, 2508)	(143, 272)
	2, 3, 3, 3, 4	5	(8224, 8281)	(185, 363)
	3, 4, 4, 4	4	(3380, 3507)	(166, 307)
	5	3	(6442, 6445)	(187, 318)
18	2	9	(3348, 3352)	(169, 257)
	3	6	(2910, 2917)	(175, 339)
	6	3	(12754, 12756)	(255, 399)
	9	2	Not Available	(277, 412)

Table 5.1 (b) Comparison of the rate of convergence of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P , the probability defined by (2.5), is 0.9 and δ , the probability defined by (5.1), is 0.5

Notation : Same as Table 5.1(a).

Figure 5.1 Plot of the average number of iterations required for convergence for the BAM and the SMA as the number of objects increases. In each case the number of classes is three.

No. Objects	Objects/Class	No. Classes	Accuracy BAM	Accuracy SMA
4	2	2	100%	87%
6	2	3	100%	82%
	2, 4	2	100%	84%
	3	2	100%	84%
9	3	3	100%	80%
	2, 3, 4	3	100%	78%
	4, 5	2	100%	82%
12	2	6	100%	63%
	2, 2, 2, 3, 3	5	100%	73%
	3	4	100%	78%
	2, 3, 3, 4	4	96%	79%
	4	3	100%	76%
	6	2	81%	80%
15	3	5	100%	77%
	2, 3, 3, 3, 4	5	100%	72%
	3, 4, 4, 4	4	100%	76%
	5	3	100%	80%
18	2	9	100%	33%
	3	6	100%	60%
	6	3	100%	67%
	9	2	Not Available	79%

Table 5.2 (a) Comparison of the accuracy of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P , the probability defined by (2.5), is 0.9 and δ , the probability defined by (5.1), is 0.25

No. Objects	Objects/Class	No. Classes	Accuracy BAM	Accuracy SMA
4	2	2	100%	87%
6	2	3	100%	82%
	2, 4	2	100%	83%
	3	2	100%	84%
9	3	3	100%	77%
	2, 3, 4	3	100%	73%
	4, 5	2	100%	88%
12	2	6	100%	53%
	2, 2, 2, 3, 3	5	100%	67%
	3	4	100%	80%
	2, 3, 3, 4	4	96%	80%
	4	3	100%	78%
	6	2	81%	83%
15	3	5	100%	69%
	2, 3, 3, 3, 4	5	100%	71%
	3, 4, 4, 4	4	100%	72%
	5	3	100%	87%
18	2	9	100%	19%
	3	6	100%	68%
	6	3	100%	75%
	9	2	Not Available	75%

Table 5.2 (b) Comparison of the accuracy of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P , the probability defined by (2.5), is 0.9 and δ , the probability defined by (5.1), is 0.5.

Figure 5.2 Plot of the accuracy of the BAM and the SMA as a function of the number of objects. In each case the number of classes is three.

P	Accuracy BAM	Accuracy SMA	
		$\delta = 0.25$	$\delta = 0.5$
0.9	100%	82%	82%
0.85	100%	80%	74%
0.8	100%	71%	63%
0.75	100%	47%	43%
0.7	100%	40%	31%
0.65	100%	28%	24%
0.6	100%	14%	5%

Table 5. Comparison of the accuracy of the BAM and the SMA as the joint access probability P decreases. In each case, the number of objects is six and the number of classes is three.

Figure 5.3 Plot of the accuracy of the BAM and the SMA as the joint access probability P decreases. In each case the number of objects is six and the number of classes is three.

5.4.7 Complexity of the SMA

If T_{SMA} queries are processed before the SMA algorithm converges, then the time complexity of the SMA is $O(T_{SMA})$. Since an SMA automaton is required to possess W actions if W objects are given, the space complexity is $O(S_{SMA})$, where S_{SMA} is the number of objects W .

Based on the experimental results we have obtained, we conjecture that

$$O(T_{SMA}) < O(T_{BAM}) \quad \text{and}$$

$$O(S_{SMA}) = O(S_{BAM}).$$

5.5 Comments

Although the SMA converges much faster than the BAM in general, it does not guarantee good accuracy. The inaccuracy of this solution cannot be blamed on the initialization process, because only one object is assigned to each action initially. However, it is clearly due to an imperfect way of moving objects around in the automaton, especially in the way the objects change actions. Consider the movements of the objects O_i and O_j , which are in states ξ_i and ξ_j , respectively. If neither ξ_i nor ξ_j is a boundary state, then no change of action will be required regardless of the response from the environment. On the other hand, if either ξ_i or ξ_j or both are boundary states, then upon receiving a penalty from the environment, a change of action may result. But we know that the environment provides "misleading" information with probability $1-P$. Although the objects are required to move only with probability δ , the probability of moving incorrectly is still finite. Thus ultimately they still possess a high probability of moving incorrectly because of the following theorem.

Theorem 5.4

Let $\mathcal{A}$ be an event that occurs with probability ϵ . If independent and identical experiments are performed, then, as time tends toward infinity, the probability of $\mathcal{A}$ occurring tends to 1 if $\epsilon > 0$. ♦ ♦ ♦

Therefore, independent of how small the probability of a "wrong" move is, the SMA will fail if the process is run long enough. This supports the degraded response of the SMA. For example, when $W = 18$ and $R = 9$, 81 per cent of the experiments yielded an erroneous partitioning. Clearly the SMA cannot be considered as an acceptable solution.

5.6 Concluding Remarks

In this chapter we have studied the solutions of the OPP by generalizing the three automata solutions to the EPP that we had proposed in the last chapter. Among these three solutions, although the Tsetlin and the Krinsky automata are easy to generalize and are capable of solving the OPP, they are feasible only when the number of objects to be partitioned is very small. On the other hand, the OMA solution is not easy to generalize for the OPP, primarily because the structure of this automaton is designed for equal-size partitioning.

In addition, a new FSSA called the Stochastic Move Automaton (SMA) has been proposed. The SMA is a reasonable solution to the OPP only if the query system is very informative (i.e., P is high). Besides this, the study of the SMA, which involved a fair bit of the time spent on this thesis, is worthwhile only because it taught us that answers of solutions involving fixed structure stochastic automata were rather fertile. Certainly it served as a stepping stone to the VSSA model that is discussed in the next chapter.

Chapter 6

A VSSA Approach To The OPP

6.1 Motivation

In the process of searching for an improvement on the SMA, the objects moving around in the automaton were examined carefully with the hope that we could come up with a recovery scheme (i.e., a scheme which, when something went wrong, led back to the point where the process started going wrong). But before such a recovery scheme can be devised, we must be able to first detect that a given situation is not correct. In order to do this, some properties of the objects must be identified, especially of the objects which lead to erroneous actions.

Let O_i , O_j and O_k be three distinct abstract objects in actions α_p , α_q and α_r , respectively and in states ξ_i , ξ_j and ξ_k , respectively. Suppose the physical pair (A_i, A_j) is accessed more frequently than the pairs (A_i, A_k) and (A_j, A_k) . Then, consider the situation when O_i is in the boundary state of α_p (i.e., $\xi_i = \lceil \alpha_p / N \rceil * N$ assuming there are N states associated with each action) while O_j and O_k are not in the boundary states of their actions. If the next query accesses the pair (A_i, A_k) and O_i decides to move, it will move from the boundary state of α_p to the boundary state of α_r , according to the transition map of the SMA. In other words, ξ_i will be updated to have the value $\lceil \alpha_r / N \rceil * N$. Since A_i and A_k are seldom accessed together, O_i is considered as having made a wrong move. As the queries are being processed, we observe that there are some special properties associated with O_i which can be described as follows :

- (1) Although O_i is brought into α_r by O_k , the probability of having the corresponding physical pair (A_i, A_k) accessed together is very small.

- (2) Although there may be other objects in α_r , most of them (if not all of them) are seldom accessed together with O_i . In other words, ideally O_i will not have any "partners" in α_r .
- (3) When O_i is accessed together with another object, say O_j , then both of them will probably be penalized, because there is a high probability that O_j will not be in α_r . Similarly, receiving a reward is unlikely.

The first property does not seem very useful unless we have frequency counters on all pairs of accessed objects. However, the use of such counters is discouraged for it defeats the purpose of learning. The second property is not useful either, for it is not reasonable to determine whether an object is in a correct action based on the number of "partners" it has in the same action. However, the last property does provide some useful information because frequent penalties received by O_i indicates that O_i may be in a wrong action. Thus a departure from that action may become necessary (each penalty brings O_i closer to the boundary state of the current action that it chooses). If an object accessed together with O_i (say O_j), did not move accidentally into the action currently chosen by O_i (say α_p), then O_j would be able to move out of α_p . However, if O_j reached the boundary state first and decided to move to α_p , it would be much harder for **both** of them to leave α_p .

In fact, underlying this property is an important concept which we informally state: "the longer O_i stays in an erroneous action α_p , the more trouble it creates". Thus, if O_i accidentally moves into a wrong action but is able to leave that action right away, then, not only will it be able to recover from the wrong transition, but it will also prevent other objects from entering that action when they are accessed together with O_i . The discovery of this important property motivated the use of **variable structure** stochastic automata,

primarily because these automata allow objects to use **two different** actions at **two consecutive** time instants. Thus, even if an object accidentally chooses a wrong action at the current instant, it is possible for it to choose another action at the next instant.

In the following sections, we shall propose a new VSSA called the Modified Linear Reward-Penalty (ML_{RP}) automaton. This automaton employs a linear updating scheme similar to the Linear Reward-Penalty (L_{RP}) scheme. Due to the similarity between the L_{RP} scheme and the one used in the ML_{RP} automaton, we shall briefly review the L_{RP} scheme before we proceed to the discussion of the ML_{RP} automaton.

6.2 The Linear Reward-Penalty Scheme

As described in [13], the Linear Reward-Penalty (L_{RP}) scheme is the earliest known updating scheme. It is a linear scheme because the $p(n+1)$ term is a linear function of the components of $p(n)$. By using the notation defined in Section 3.6.2, this scheme can be obtained by setting

$$g_i(P) = a * p_i \quad \text{and} \quad h_i(P) = [b/(R-1)] - b * p_i$$

Thus, the L_{RP} scheme is defined as :

$$\begin{aligned} p_i(n+1) &= p_i(n) + \sum_{j \neq i} a * p_j(n) \\ &= p_i(n) + a * (1 - p_i(n)) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\ p_i(n+1) &= p_i(n) - a * p_i(n) \\ &= (1 - a) * p_i(n) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\ p_i(n+1) &= p_i(n) - \sum_{j \neq i} [b/(R-1)] - b * p_j(n) \\ &= p_i * (1 - b) && \text{if } \alpha_i \text{ is chosen and } \beta = 1 \\ p_i(n+1) &= p_i(n) + [b/(R-1)] - b * p_i(n) \\ &= [b/(R-1)] + (1 - b) * p_i(n) && \text{if } \alpha_i \text{ is chosen and } \beta = 1 \end{aligned}$$

If there are R actions in the action set, then $1 \leq i, j \leq R$.

The L_{RP} scheme has been extensively studied in the literature. It has been shown that the scheme is ergodic and that an automaton using the L_{RP} scheme is expedient in all stationary random environments.

6.3 The Modified Linear Reward-Penalty Scheme

6.3.1 Motivating Idea

Just like the OMA and the SMA, the objects to be partitioned are represented by a set of abstract objects which are allowed to move around in the automaton. The motivation behind the ML_{RP} scheme is as follows: when a pair of objects (A_i, A_j) is accessed and if O_i and O_j choose the same action, say α_k , then they will be rewarded by the interacting environment. Just like the L_{RP} scheme, this is achieved by **increasing** the probability of them choosing α_k and at the same time **decreasing** the probabilities of them choosing **all other** actions. On the other hand, if O_i and O_j choose actions α_p and α_q respectively, where $\alpha_p \neq \alpha_q$, then they will be penalized. Since it **may be** the case that A_i and A_j should be in the same class, both O_i and O_j should "try" to use a common action. In such a case, instead of letting O_i move to α_q and O_j move to α_p , we modify the probabilities so that O_i chooses α_q and O_j chooses α_p with higher probabilities. Since the other actions α_k (for $k \neq i, j$) are not involved, the probabilities of them being chosen by O_i and O_j should not be changed. To be specific, since α_p is considered as an inappropriate action for O_i and α_q is the one that O_i should try, p_{ip} (the probability of O_i choosing α_p) should be decremented. Consequently, the decrement should be added to p_{iq} (the probability of O_i choosing α_q). Similarly, the value of p_{jp} should be decreased and the value of p_{jp} should be increased by the corresponding amount.

However, since we are not sure whether the accessed pair (A_i, A_j) should really be placed in the same class, the amount of change upon receiving a penalty should be **small** so that a recovery due to an erroneous choice can be more easily achieved.

6.3.2 Structure of the ML_{RP} Automaton

If $\{A_1, \dots, A_W\}$ is the set of W given objects, then just like the SMA, there will be W actions in the ML_{RP} automaton with each action representing a certain class. By using the notation defined in Section 3.6.1, the ML_{RP} can be defined as a quadruple (α, β, P, F) where

- (i) $\alpha = \{\alpha_1, \dots, \alpha_W\}$ is the set of actions that the abstract objects must choose from.
- (ii) $\beta = \{0, 1\}$ is the set of inputs. The value '0' represents a reward, '1' is a penalty.
- (iii) $P = \{P_1, \dots, P_W\}$ is the set of action probability vectors such that $P_i = [p_{i1}, \dots, p_{iW}]^T$, for all $1 \leq i \leq W$. Each component p_{ij} , $1 \leq j \leq W$, represents the probability that action α_j will be chosen by O_i . Also, $\sum_{1 \leq j \leq W} p_{ij} = 1.0$
- (iv) F is the probability updating scheme. If A_i and A_j are the accessed objects and their abstract counterparts choose actions α_m and α_n , respectively, $1 \leq m, n \leq W$, then the ML_{RP} scheme is defined as follows, for $0 < \lambda_1, \lambda_2 < 1$.

$$\begin{cases} p_{im}(n+1) = 1 - \lambda_1 * (1 - p_{im}(n)) & \text{if } \beta = 0 \text{ (i.e., } \alpha_m = \alpha_n) \\ p_{ik}(n+1) = \lambda_1 * p_{ik}(n) & 1 \leq k \leq W \text{ and } k \neq i \end{cases}$$

$$\begin{cases} p_{in}(n+1) = p_{in}(n) + (1 - \lambda_2) * p_{im}(n) & \text{if } \beta = 1 \text{ (i.e., } \alpha_m \neq \alpha_n) \\ p_{im}(n+1) = \lambda_2 * p_{im}(n) \end{cases}$$

The probabilities $\{p_{jk}\}$ are updated in an analogous manner.

6.3.3 Environment

Just like the SMA discussed in last chapter, the "query-system" is the environment which the ML_{RP} automaton interacts with. The input to it is the set of actions chosen by the abstract objects. Every time a pair of objects (A_i, A_j) is accessed, the environment compares the actions that are chosen by the abstract objects, say α_p for O_i and α_q for O_j , and sends out a response according to the following rules :

- (1) If $\alpha_p = \alpha_q$, then a reward (i.e., $\beta = 0$), will be sent out.
- (2) If $\alpha_p \neq \alpha_q$, then a penalty (i.e., $\beta = 1$), will be sent out.

6.3.4 Implementation Details

If $\{O_1, \dots, O_W\}$ is the set of W abstract objects, then there will be W actions $\{\alpha_1, \dots, \alpha_W\}$ available for each of these objects. Since each object can be in any of the W actions, there is an action probability vector $[p_{i1}, \dots, p_{iW}]^T$ associated with object O_i , for $1 \leq i \leq W$, such that the value p_{ik} represents the probability that O_i will choose action α_k at a given instant. Just like the SMA, each object O_i is initially assigned to choose an action α_i . Thus, if O_i and O_j are two distinct objects, then they will not be initially assigned to the same action. With no loss of generality, we can assume that O_i is assigned to choose α_i (i.e., $p_{ii} = 1.0$ and $p_{ij} = 0.0$ for $1 \leq i, j \leq W$ and $i \neq j$).

When a pair of objects (A_i, A_j) is accessed, if the corresponding abstract objects choose actions α_p and α_q , respectively, then they will be rewarded if $\alpha_p = \alpha_q$; otherwise, they will be penalized. Upon receiving a reward, both the values of p_{ip} and p_{jq} will be increased while all other components of the action probability vectors associated with O_i and O_j will be correspondingly decreased according to the ML_{RP} scheme. In the case of receiving a penalty, the values of

p_{iq} and p_{jp} will be increased and only p_{ip} and p_{jq} will be decreased.

After T accesses, the physical objects will be partitioned based on the actions chosen by the abstract objects. For the sake of completeness, we present below an algorithmic version of the ML_{RP} automaton.

Algorithm ML_{RP} Automaton

Input : The abstract objects $\{O_1, \dots, O_W\}$, the reward parameter λ_1 and the penalty parameter λ_2 . Also coming into the algorithm is a stream of queries (A_i, A_j) .

Output : A periodic partitioning of the objects.

Notation : α_p and α_q are the actions chosen by O_i and O_j at a given instant. They are integers between 1 and W . p_{ij} is the probability that O_i chooses group j at a given instant, $0 \leq p_{ij} \leq 1.0$, $\sum_{1 \leq j \leq W} p_{ij} = 1.0$

Method : With no loss of generality, initialize $p_{ii} = 1.0$ and $p_{ij} = 0.0$.

Repeat

For a sequence of T queries **do**

 ReadQuery (A_i, A_j)

If $(\alpha_p = m \text{ and } \alpha_q = m)$ (* Reward *)

$p_{im} = 1 - \lambda_1 (1 - p_{im})$

$p_{jm} = 1 - \lambda_1 (1 - p_{jm})$

For $(k = 1 \text{ to } W, k \neq m)$

$p_{ik} = \lambda_1 p_{ik}$

$p_{jk} = \lambda_1 p_{jk}$

EndFor

EndIf

If $(\alpha_p = m \text{ and } \alpha_q = n)$ (* Penalty *)

$p_{in} = p_{in} + (1 - \lambda_2) p_{im}$

$p_{im} = \lambda_2 p_{im}$

$p_{jn} = p_{jn} + (1 - \lambda_2) p_{jm}$

$p_{jm} = \lambda_2 p_{jm}$

EndIf

EndFor

 Print out partitions based on the probabilities p_{ij}

ForEver

End Algorithm ML_{RP}

6.3.5 An ML_{RP} Example

To illustrate how the ML_{RP} works, let us consider an example of partitioning a set of four objects $\{A_1, A_2, A_3, A_4\}$ into two classes. Suppose the queries against these objects request the pairs (A_1, A_2) and (A_3, A_4) more frequently than any other pairs. In other words, the desired result is that A_1 and A_2 should be found in the same class and A_3 and A_4 in another class. Besides, in order to render the example brief, we assume that λ_1 and λ_2 are set to 0.1 and 0.95 respectively.

In this example, we denote the action probability vector associated with abstract object O_i as $P_i = [p_{i1}, p_{i2}, p_{i3}, p_{i4}]^T$, where p_{ij} is the probability of O_i choosing α_j , for all $1 \leq i, j \leq 4$. From step 1 through step 7, the vectors to the left hand side of the ' $\rightarrow$ ' represent the action probabilities before processing the query; the right hand side vectors represent the values after processing the query. The algorithm is considered to have converged, if $p_{ij} \geq 0.95$, for all $1 \leq i \leq 4$, and some $1 \leq j \leq 4$.

The set of input queries is as follows :

Query 1 : Access objects A_1 and A_2 .

Query 2 : Access objects A_1 and A_4 .

Query 3 : Access objects A_3 and A_4 .

Query 4 : Access objects A_1 and A_2 .

Query 5 : Access objects A_3 and A_4 .

Query 6 : Access objects A_1 and A_2 .

Query 7 : Access objects A_3 and A_4 .

Based on these queries, A_1 and A_2 should be in one class and A_3 and A_4 should be in another.

Step 0 : Initially O_i is assigned to α_i (i.e. $p_{ii} = 1.0$ and $p_{ij} = 0.0$), for all $1 \leq i \leq 4$.

$$\begin{aligned} P_1 &: [1.0000 \ 0.0000 \ 0.0000 \ 0.0000]^T \\ P_2 &: [0.0000 \ 1.0000 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \\ P_4 &: [0.0000 \ 0.0000 \ 0.0000 \ 1.0000]^T \end{aligned}$$

Step 1 : The pair (A_1, A_2) is accessed. If O_1 chooses α_1 and O_2 chooses α_2 , both will be penalized by decreasing the values of p_{11} and p_{22} from 1.0 to 0.95 and increasing the values of p_{12} and p_{21} from 0.0 to 0.05.

$$\begin{aligned} P_1 &: [1.0000 \ 0.0000 \ 0.0000 \ 0.0000]^T \rightarrow [0.9500 \ 0.0500 \ 0.0000 \ 0.0000]^T \\ P_2 &: [0.0000 \ 1.0000 \ 0.0000 \ 0.0000]^T \rightarrow [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \rightarrow [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \\ P_4 &: [0.0000 \ 0.0000 \ 0.0000 \ 1.0000]^T \rightarrow [0.0000 \ 0.0000 \ 0.0000 \ 1.0000]^T \end{aligned}$$

Step 2 : The pair (A_1, A_4) is accessed. If O_1 chooses α_1 and O_4 chooses α_4 , as in Step 1, both will be penalized by decreasing the value of p_{11} from 0.95 to 0.925 and p_{44} from 1.0 to 0.95.

$$\begin{aligned} P_1 &: [0.9500 \ 0.0500 \ 0.0000 \ 0.0000]^T \rightarrow [0.9025 \ 0.0500 \ 0.0000 \ 0.0475]^T \\ P_2 &: [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \rightarrow [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \rightarrow [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \\ P_4 &: [0.0000 \ 0.0000 \ 0.0000 \ 1.0000]^T \rightarrow [0.0500 \ 0.0000 \ 0.0000 \ 0.9500]^T \end{aligned}$$

Step 3 : The pair (A_3, A_4) is accessed. If O_3 chooses α_3 and O_4 chooses α_4 , then both will be penalized by decreasing the value of p_{33} from 1.0 to 0.95 and p_{44} from 0.95 to 0.9025.

$$\begin{aligned} P_1 &: [0.9025 \ 0.0500 \ 0.0000 \ 0.0475]^T \rightarrow [0.9025 \ 0.0500 \ 0.0000 \ 0.0475]^T \\ P_2 &: [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \rightarrow [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 1.0000 \ 0.0000]^T \rightarrow [0.0000 \ 0.0000 \ 0.9500 \ 0.0500]^T \\ P_4 &: [0.0500 \ 0.0000 \ 0.0000 \ 0.9500]^T \rightarrow [0.0500 \ 0.0000 \ 0.0475 \ 0.9025]^T \end{aligned}$$

Step 4 : The pair (A_1, A_2) is accessed. If O_1 chooses α_1 and O_2 chooses α_1 , then both will be rewarded by increasing the value of p_{11} from 0.9025 to 0.9903 and p_{21} from 0.05 to 0.905.

$$\begin{aligned} P_1 &: [0.9025 \ 0.0500 \ 0.0000 \ 0.0475]^T \rightarrow [0.9903 \ 0.0050 \ 0.0000 \ 0.0047]^T \\ P_2 &: [0.0500 \ 0.9500 \ 0.0000 \ 0.0000]^T \rightarrow [0.9050 \ 0.0950 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 0.9500 \ 0.0500]^T \rightarrow [0.0000 \ 0.0000 \ 0.9500 \ 0.0500]^T \\ P_4 &: [0.0500 \ 0.0000 \ 0.0475 \ 0.9025]^T \rightarrow [0.0500 \ 0.0000 \ 0.0475 \ 0.9025]^T \end{aligned}$$

Step 5 : The pair (A_3, A_4) is accessed. If O_3 chooses α_3 and O_4 chooses α_4 , then both will be rewarded by increasing the value of p_{33} from 0.95 to 0.995 and p_{43} from 0.472 to 0.9048.

$$\begin{aligned} P_1 &: [0.9903 \ 0.0050 \ 0.0000 \ 0.0047]^T \rightarrow [0.9903 \ 0.0050 \ 0.0000 \ 0.0047]^T \\ P_2 &: [0.9050 \ 0.0950 \ 0.0000 \ 0.0000]^T \rightarrow [0.9050 \ 0.0950 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 0.9500 \ 0.0500]^T \rightarrow [0.0000 \ 0.0000 \ 0.9950 \ 0.0050]^T \\ P_4 &: [0.0500 \ 0.0000 \ 0.0475 \ 0.9025]^T \rightarrow [0.0050 \ 0.0000 \ 0.9048 \ 0.0902]^T \end{aligned}$$

Step 6 : The pair (A_1, A_2) is accessed. If O_1 chooses α_1 and O_2 chooses α_1 , then both will be rewarded by increasing the value of p_{11} from 0.9903 to 0.999 and p_{21} from 0.905 to 0.9905.

$$\begin{aligned} P_1 &: [0.9903 \ 0.0050 \ 0.0000 \ 0.0047]^T \rightarrow [0.9990 \ 0.0005 \ 0.0000 \ 0.0005]^T \\ P_2 &: [0.9050 \ 0.0950 \ 0.0000 \ 0.0000]^T \rightarrow [0.9905 \ 0.0095 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 0.9950 \ 0.0050]^T \rightarrow [0.0000 \ 0.0000 \ 0.9950 \ 0.0050]^T \\ P_4 &: [0.0050 \ 0.0000 \ 0.9048 \ 0.0902]^T \rightarrow [0.0050 \ 0.0000 \ 0.9048 \ 0.0902]^T \end{aligned}$$

Step 7 : The pair (A_3, A_4) is accessed. If O_3 chooses α_3 and O_4 chooses α_3 , then both will be rewarded by increasing the value of p_{33} from 0.995 to 0.9995 and p_{43} from 0.9048 to 0.9905.

$$\begin{aligned} P_1 &: [0.9990 \ 0.0005 \ 0.0000 \ 0.0005]^T \rightarrow [0.9990 \ 0.0005 \ 0.0000 \ 0.0005]^T \\ P_2 &: [0.9905 \ 0.0095 \ 0.0000 \ 0.0000]^T \rightarrow [0.9905 \ 0.0095 \ 0.0000 \ 0.0000]^T \\ P_3 &: [0.0000 \ 0.0000 \ 0.9950 \ 0.0050]^T \rightarrow [0.0000 \ 0.0000 \ 0.9995 \ 0.0005]^T \\ P_4 &: [0.0050 \ 0.0000 \ 0.9048 \ 0.0902]^T \rightarrow [0.0005 \ 0.0000 \ 0.9905 \ 0.0090]^T \end{aligned}$$

After step 7, since for each P_i , one of the $p_{ij} \geq 0.95$, $1 \leq j \leq 4$, the algorithm is considered to have converged, and the desired partitioning is obtained.

6.3.6 Choice of the Parameters

The two parameters involved in the ML_{RP} automaton are λ_1 and λ_2 , where λ_1 is the one involved in updating the action probability vector when a reward is received and λ_2 is the one involved in such a change when a penalty is received. Since the ML_{RP} scheme is of the form $p_i = \lambda_1 * p_i$ if $\beta = 0$, and $p_i = \lambda_2 * p_i$ if $\beta = 1$, intuitively, the values of both λ_1 and λ_2 should be close to unity

so that on receiving a reward or a penalty the changes made in the action probabilities is not substantial.

6.3.6.1 Experimental Results for the Parameters

To date, the analysis of the ML_{RP} automaton remains unsolved. As it stands now, the most suitable parameters are chosen experimentally. The ML_{RP} automaton discussed above was simulated to partition a set of six objects into three equal-size classes with different values of λ_1 and λ_2 in order to observe the effect of these parameters on the performance of the automaton.

6.3.6.1.1 Experiments Performed In an Informative Environment

The ML_{RP} was first simulated in an informative environment (i.e., the P value defined by (2.5) was set to 0.9). In this case, λ_1 was chosen from the interval $[0.025, 0.8]$ and λ_2 was set to 0.975, 0.95, 0.9 and 0.85. A summary of the results that have been obtained is found in Tables 6.1 and 6.2. From Table 6.1, we observe that both λ_1 and λ_2 have control over the rate of convergence of the ML_{RP} automaton. For example, when $\lambda_1 = 0.6$, the ML_{RP} automaton converged after 67 iterations if $\lambda_2 = 0.975$. The corresponding figure for $\lambda_2 = 0.85$ is 112. This is a difference of 45 iterations. On the other hand, if $\lambda_2 = 0.975$, the ML_{RP} automaton converged after 37 iterations when $\lambda_1 = 0.025$ and converged after 140 iterations when $\lambda_1 = 0.8$; a difference of 103 iterations. The effect of both λ_1 and λ_2 on the rate of convergence of the ML_{RP} automaton is obvious. Figure 6.1 graphically compares the rate of convergence of the ML_{RP} automaton as a function of λ_1 , for different values of λ_2 .

Experimentally, both λ_1 and λ_2 affect the accuracy of the scheme. From Table 6.2, we observe that for $\lambda_1 = 0.6$, the ML_{RP} automaton gave 99 per cent

accuracy when $\lambda_2 = 0.975$. This figure decreased to only 65 per cent when $\lambda_2 = 0.85$. On the other hand, when $\lambda_2 = 0.9$, the automaton gave 96 per cent accuracy when $\lambda_1 = 0.025$. But the accuracy decreased to 31 per cent when $\lambda_1 = 0.8$. However, the effect of λ_1 on the accuracy of the scheme decreased as λ_2 was increased. Figure 6.2 compares the accuracy of the ML_{RP} automaton as a function of λ_1 , for different values of λ_2 .

Based on the above observations, we conjecture the following :

- (a) In general, the rate of convergence and the accuracy of the ML_{RP} automaton are controlled by both λ_1 and λ_2 .
- (b) When λ_2 is large, λ_1 does not affect the accuracy of the automaton.

λ_1	Converging Rate ML_{RP}			
	$\lambda_2 = 0.975$	$\lambda_2 = 0.95$	$\lambda_2 = 0.9$	$\lambda_2 = 0.85$
0.025	(36, 37)	(27, 28)	(22, 23)	(21, 22)
0.05	(36, 37)	(27, 28)	(22, 23)	(21, 22)
0.1	(38, 42)	(30, 34)	(28, 30)	(28, 30)
0.2	(41, 42)	(34, 35)	(30, 31)	(31, 32)
0.3	(43, 47)	(36, 41)	(37, 39)	(39, 41)
0.4	(48, 52)	(41, 45)	(44, 46)	(51, 53)
0.5	(54, 58)	(48, 52)	(57, 61)	(71, 74)
0.6	(63, 67)	(60, 65)	(82, 86)	(108, 112)
0.7	(80, 88)	(85, 93)	(148, 154)	(245, 251)
0.8	(126, 140)	(191, 204)	(570, 579)	(739, 749)

Table 6.1 Comparison of the converging rate of the ML_{RP} automaton for various values of λ_1 and λ_2 . In this case, the joint access probability is 0.9, and six objects are partitioned into three classes.

λ_1	Accuracy ML_{RP}			
	$\lambda_2 = 0.975$	$\lambda_2 = 0.95$	$\lambda_2 = 0.9$	$\lambda_2 = 0.85$
0.025	99%	99%	96%	94%
0.05	99%	99%	97%	94%
0.1	100%	98%	95%	89%
0.2	100%	98%	94%	87%
0.3	99%	100%	94%	88%
0.4	99%	100%	94%	87%
0.5	100%	100%	93%	77%
0.6	99%	100%	94%	65%
0.7	100%	100%	83%	35%
0.8	100%	97%	31%	0%

Table 6.2 Comparison of the accuracy of the ML_{RP} automaton for various values of λ_1 and λ_2 . In this case, the joint access probability is 0.9 and six objects are partitioned into three classes.

Figure 6.1 Comparison of the rate of convergence of the ML_{RP} automaton, as a function of λ_1 , for $\lambda_2 = 0.975, 0.95, 0.9, 0.85$. In this case, for each point on the graph the number of objects is six, the number of classes is three, and the joint access probability is 0.9.

Figure 6.2 Comparison of the accuracy of the ML_{RP} automaton as a function of λ_1 , for $\lambda_2 = 0.975, 0.95, 0.9, 0.85$. In this case, for each point on the graph the number of objects is six, the number of classes is three, and the joint access probability is 0.9.

6.3.6.1.2 Experiments Performed In Less Informative Environments

In order to determine suitable values for λ_1 , the ML_{RP} automaton was further simulated to find the partitioning of the objects in various environments (i.e., by using various values of the joint access probability P defined by (2.5)). In this case again, the automaton was simulated to partition a set of six objects into three equal-size classes. The values of λ_1 were set to 0.1, 0.5 and 0.8 to represent small, medium and large values, respectively. A summary of the results that have been obtained is given in Tables 6.3 and 6.4.

From Table 6.3, we observe that the ML_{RP} automaton yielded an excellent rate of convergence when $\lambda_1 = 0.1$. In this case, the automaton converged to the correct partitioning after 44 iterations when $P = 0.9$ and after 75 iterations when $P = 0.6$. Similarly, when $\lambda_1 = 0.5$, the automaton converged after 60 iterations when $P = 0.9$ and after 131 iterations when $P = 0.6$. However, when $\lambda_1 = 0.8$, it took the automaton 140 iterations to converge to the correct partitioning when $P = 0.9$, and this figure increased to 549 when P decreased to 0.6. Figure 6.3 compares the converging rate of the ML_{RP} automaton when $\lambda_2 = 0.975$ and when $\lambda_1 = 0.1, 0.5$ and 0.8 . From the graph, we observe that in general for small values of λ_1 , the rate of convergence of the automaton decreased (the number of iterations required increased) gradually when P decreased. On the other hand, when λ_1 was large, the rate of convergence decreased sharply when P was decreased.

As far as accuracy is concerned, we observe from Table 6.4 that the automaton demonstrates excellent accuracy when $P = 0.9$, for the values of λ_1 investigated. As P decreased (i.e. the information contained in the queries decreased), the accuracy of the automaton decreased gradually if λ_1 was either 0.1 or 0.5. Thus, when $P = 0.85$, the automaton gave 96 per cent and 98

per cent accuracy for $\lambda_1 = 0.1$ and 0.5 , respectively; when $P = 0.65$, the figures recorded are 73 per cent and 77 per cent, respectively. On the other hand, if λ_1 was large (i.e., $\lambda_1 = 0.8$), the accuracy of the automaton decreased sharply as P decreased. In this case, the automaton gave 99 per cent accuracy when $P = 0.85$. This figure decreased to only 11 per cent when $P = 0.65$. Figure 6.4 graphically compares the accuracy of the ML_{RP} automaton for $\lambda_2 = 0.975$ and various values of λ_1 , namely 0.1 , 0.5 , and 0.8 . For all the experiments performed, we observed that if the environment was characterized by a probability P , the probability of the automaton yielding 100 per cent accuracy is in general at least P . Further we observed that, in general, the automaton gave better accuracy when $\lambda_1 = 0.5$ than when $\lambda_1 = 0.1$ and 0.8 .

6.3.6.2 Comments

From the experimental evidence, we conclude that λ_2 should be large (i.e., as close to unity as possible), if an accurate performance is to be demonstrated. In the case of λ_1 , although small values of λ_1 yield faster scheme, experimental results suggest that, in general, values in the neighbourhood of 0.5 yield more accurate results.

P	Converging Rate ML_{RP}		
	$\lambda_1 = 0.8$	$\lambda_1 = 0.5$	$\lambda_1 = 0.1$
0.9	(126, 140)	(56, 60)	(39, 44)
0.85	(151, 174)	(60, 65)	(44, 48)
0.8	(225, 246)	(67, 72)	(48, 52)
0.75	(340, 366)	(71, 78)	(51, 55)
0.7	(586, 603)	(88, 95)	(60, 64)
0.65	(686, 699)	(94, 102)	(64, 67)
0.6	(537, 549)	(123, 131)	(71, 75)

Table 6.3 Comparison of the rate of convergence of the ML_{RP} automaton for different λ_1 , for various joint access probabilities P defined by (2.5). In this case, $\lambda_2 = 0.975$ and six objects are partitioned into three classes.

P	Accuracy ML_{RP}		
	$\lambda_1 = 0.8$	$\lambda_1 = 0.5$	$\lambda_1 = 0.1$
0.9	100%	100%	100%
0.85	99%	98%	96%
0.8	94%	98%	97%
0.75	82%	89%	87%
0.7	55%	85%	82%
0.65	11%	77%	73%
0.6	1%	54%	56%

Table 6.4 Comparison of the accuracy of the ML_{RP} automaton for different λ_1 , for various joint access probabilities P defined by (2.5). In this case, $\lambda_2 = 0.975$ and six objects are partitioned into three classes.

Figure 6.3 Plot of the rate of convergence of the ML-PP automaton as a function of the joint access probability P defined by (2.5), for different values of λ_1 . In this case, $\lambda_2 = 0.975$ and for each point on the graph the number of objects is six and the number of classes is three.

Figure 6.4 Plot of the accuracy of the ML_{RP} automaton as a function of the joint access probability P defined by (2.5), for different values of λ_1 . In this case, $\lambda_2 = 0.975$ and for each point on the graph the number of objects is six and the number of classes is three.

6.3.7 Experimental Comparison with the BAM

The ML_{RP} discussed above was simulated and compared with the BAM primarily to:

- (a) compare the rate of convergence and the accuracy of the ML_{RP} automaton with the BAM as a function of the number of objects to be partitioned,
- (b) compare the rate of convergence and the accuracy of the ML_{RP} automaton with the BAM as a function of the joint access probability defined by (2.5)

In each set up, one hundred experiments were performed in order to obtain accurate results.

6.3.7.1 Initialization

If W objects were given, then initially each object O_i was assigned arbitrarily to an action α_j , for $1 \leq i, j \leq W$, with probability 1 (i.e., $p_{ij} = 1.0$ and $p_{ik} = 0.0$ for all $1 \leq k \leq W$ and $k \neq j$).

The initial set up of the BAM was the same as that described in Section 2.2.7.1.

6.3.7.2 Parameters and Assumptions

The two parameters involved in the ML_{RP} were λ_1 and λ_2 . Based on the discussion in Section 6.3.6, these values were assigned as follows:

- (1) λ_1 and λ_2 were set to 0.5 and 0.975 respectively in all the experiments performed.
- (2) when the solutions were compared as a function of the number of objects, the number of objects varied from four to eighteen. In the case when the

joint access probability P defined by (2.5) was varied, the number of objects was six and the number of classes was three. In the latter case, P was chosen from the interval $[0.6, 0.9]$.

The parameters used in the case of the BAM were the same as those described in Section 2.2.7.2. The assumptions used were the same as those described in Section 2.2.7.3.

6.3.7.3 Convergence

The ML_{RP} automaton was considered to have converged if every object O_i , $1 \leq i \leq W$, moving around in the automaton chose any of the W actions with a probability of 0.95 or more. This is quite realistic because when such a situation occurs the object will not be able to change to other actions easily.

6.3.7.4 Results

A summary of the results that have been obtained is found in Tables 6.5 and 6.6. From Table 6.5, we observe the superiority of the rate of convergence of the ML_{RP} automaton to the BAM. For example, when fifteen objects were partitioned into four unequal-size classes, the BAM converged after 3507 iterations. The corresponding figure for the ML_{RP} automaton when $\lambda_1 = 0.5$ and $\lambda_2 = 0.975$ is 383. This is an improvement by a factor of about 9.1. A comparative graphical sketch of the average number of iterations required for convergence of the BAM and the ML_{RP} automaton is drawn in Figure 6.5 where objects were partitioned into three equal-size classes.

In terms of accuracy, we observe from Table 6.6 that the BAM gave excellent accuracy, namely between 81 per cent and 100 per cent with the mean of 99.1 per cent. In the case of the ML_{RP} automaton, when $\lambda_1 = 0.5$ and

$\lambda_2 = 0.975$, the accuracy ranged from 95 per cent to 100 per cent with the mean of 98.6 per cent. This implies that the BAM is about 0.5 per cent more accurate than the ML_{RP} automaton. Figure 6.6 compares the accuracy of the BAM and the ML_{RP} automaton graphically as a function of the number of objects. In this case, for each point on the graph the number of classes is three.

Table 6.7 summarizes the performance of the BAM and the ML_{RP} automaton in various environments (i.e., for various joint access probabilities P defined by (2.5)). From the table, we observe that, in general, the BAM converged more slowly than the ML_{RP} automaton. For example, when $P = 0.8$, the BAM converged after 332 iterations and the ML_{RP} automaton converged after 72 iterations. The ML_{RP} automaton demonstrated an improvement by a factor of 4.6. In terms of accuracy, we observe that the BAM provided excellent and stable accuracy (i.e., 100 per cent in the various environments). In the case of the ML_{RP} automaton, when it interacted with an environment characterized by the parameter P , in general it yielded 100 per cent accuracy with probability at least P . For example, when $P = 0.65$, the ML_{RP} automaton gave 77 per cent accuracy. Figures 6.7 and 6.8 graphically illustrate the comparison of the accuracy of the BAM and the ML_{RP} automaton as a function of the joint access probability P .

No. Objects	Objects/Class	No. Classes	# iterations BAM	# iterations ML _{RP}
4	2	2	(21, 40)	(34, 37)
6	2	3	(255, 269)	(56, 60)
	2, 4	2	(55, 70)	(157, 163)
	3	2	(243, 251)	(91, 95)
9	3	3	(745, 762)	(144, 151)
	2, 3, 4	3	(365, 377)	(249, 258)
	5	2	(711, 728)	(243, 250)
12	2	6	(1520, 1527)	(120, 130)
	2, 2, 2, 3, 3	5	(1940, 1977)	(242, 252)
	3	4	(1885, 1894)	(212, 222)
	2, 3, 3, 4	4	(8193, 8197)	(326, 342)
	4	3	(2914, 2925)	(279, 288)
	6	2	(3593, 3609)	(382, 392)
15	3	5	(2501, 2508)	(279, 291)
	2, 3, 3, 3, 4	5	(8224, 8281)	(403, 416)
	3, 4, 4, 4	4	(3380, 3507)	(371, 383)
	5	3	(6442, 6445)	(467, 478)
18	2	9	(3348, 3352)	(216, 233)
	3	6	(2910, 2917)	(322, 335)
	6	3	(12754, 12756)	(606, 619)
	9	2	Not Available	(797, 813)

Table 6.5 Comparison of the rate of convergence of the BAM and the ML_{RP} automaton solutions for the OPP. In the latter case, the values of λ_1 and λ_2 are 0.5 and 0.975, respectively. Also, the joint access probability P defined by (2.5) is 0.9.

Notation : Number of iterations is given as a pair (x, y) , where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning.

Figure 6.5 Plot of the average number of iterations required for convergence for the BAM and the ML_{RP} as the number of objects increases. In each case the number of classes is three.

No. Objects	Objects/Class	No. Classes	Accuracy BAM	Accuracy ML _{RP}
4	2	2	100%	99%
6	2	3	100%	100%
	2, 4	2	100%	95%
	3	2	100%	99%
9	3	3	100%	100%
	2, 3, 4	3	100%	99%
	4, 5	2	100%	96%
12	2	6	100%	99%
	2, 2, 2, 3, 3	5	100%	100%
	3	4	100%	99%
	2, 3, 3, 4	4	96%	100%
	4	3	100%	100%
	6	2	81%	99%
15	3	5	100%	100%
	2, 3, 3, 3, 4	5	100%	99%
	3, 4, 4, 4	4	100%	100%
	5	3	100%	99%
18	2	9	100%	100%
	3	6	100%	100%
	6	3	100%	100%
	9	2	Not Available	99%

Table 6.6 Comparison of the accuracy of the BAM and the ML_{RP} automaton solutions for the OPP. In the latter case, the values of λ_1 and λ_2 are 0.5 and 0.975, respectively. Also, the joint access probability P defined by (2.5) is 0.9.

Figure 6.6 Plot of the accuracy of the BAM and the ML_{RP} automaton as a function of the the number of objects. In each case the number of classes is three.

P	# iterations BAM	# iterations ML _{RP}	Accuracy BAM	Accuracy ML _{RP}
0.9	(255, 269)	(56, 60)	100%	100%
0.85	(272, 285)	(60, 65)	100%	98%
0.8	(322, 332)	(67, 72)	100%	98%
0.75	(341, 349)	(71, 78)	100%	89%
0.7	(375, 385)	(88, 95)	100%	85%
0.65	(414, 421)	(94, 102)	100%	77%
0.6	(493, 501)	(123, 131)	100 %	54%

• • **Table 6.7** Comparison of the converging rate and the accuracy of the BAM and the ML_{RP} automaton as a function of the joint access probability, defined by (2.5) when six objects are partitioned into three equal-size classes.

Figure 6.7 Plot of the average number of iterations required for convergence for the BAM and the ML_{RP} automaton as a function of the joint access probability P . In this case the number of objects is six and the number of classes is three.

Figure 6.8 Plot of the accuracy of the BAM and the ML_{RP} automaton as a function of the joint access probability P. In this case the number of objects is six and the number of classes is three.

6.4 Comments

The ML_{RP} scheme discussed in this chapter shows great improvement on accuracy over the SMA. Although it appears to be marginally less accurate than the BAM, the accuracy obtained by this scheme is definitely acceptable. Typically, a 95 per cent accuracy is obtained in an environment characterized by the parameter $P = 0.9$. However, in all cases, the ML_{RP} scheme is much faster than the BAM.

Experimental results suggest that λ_1 should take a value in the neighbourhood of 0.5 and λ_2 should take a value close to unity, $0 < \lambda_1, \lambda_2 < 1$. In the case of λ_2 , using a larger value (i.e., closer to unity) is quite intuitively plausible because this implies that on being penalized only small changes are made on the action probabilities. Therefore, any erroneous choice can be rectified easily. On the other hand, the fact that λ_1 should use values near 0.5 seems to contradict our intuition because this causes substantial changes on the action probabilities upon receiving rewards. This is intuitively not very desirable since the information leading to a reward may be misleading. We are unable to explain this counter-intuitive behaviour.

Although the ML_{RP} scheme provides such good results, analyzing its ϵ -optimality properties are definitely non-trivial. We believe that this involves analyzing a complex co-operative automaton game between the participating abstract objects. However, in a straightforward way, the ML_{RP} automaton can be shown to be **expedient**. We shall show this in the following theorem. Besides, experimental results seem to indicate that it is also ϵ -optimal.

Theorem 6.1

The ML_{RP} automaton is expedient if $0 < \lambda_1, \lambda_2 < 1.0$.

Proof :

Each object O_i moving around in the automaton is initially assigned to a distinct action (representing a class) α_k , $1 \leq i, k \leq W$, $P_{ik} = 1.0$ and $P_{ij} = 0.0$; for all $1 \leq j \leq W$ and $j \neq k$. With no loss of generality, we assume $i = k$. Since each O_i is in a distinct class, the probability of an accessed pair of objects being rewarded is zero. Thus, the initial expected penalty, denoted as M_0 , is unity.

Suppose at the first time instant the pair (A_i, A_j) is accessed. Since $P_{ii} = 1.0$ and $P_{jj} = 1.0$, O_i and O_j will not be able to choose the same class. Thus both O_i and O_j will be penalized. According to the ML_{RP} scheme, P_{ii} is updated to λ_2 and P_{jj} is updated to $P_{jj} + (1 - \lambda_2) * P_{ii}$. Similarly P_{ji} changes to $\lambda_2 * P_{ji}$ and P_{ij} changes to $P_{ij} + (1 - \lambda_2) * P_{ji}$. After the first query, the expected probability that O_i and O_j will be rewarded if (A_i, A_j) is accessed will become non-zero, if $0.0 < \lambda_2 < 1.0$. Thus, if we denote the expected penalty at the first time instant as $E[M(1)]$, then $E[M(1)] < M_0$.

After the first instant, independent of whether O_i and O_j are rewarded or penalized, the values of both P_{ii} and P_{jj} cannot be changed back to 1 if $0 < \lambda_1, \lambda_2 < 1$. Similarly, the values of both P_{ji} and P_{ij} cannot be changed back to 0. Therefore, the limiting expected penalty, $E[M(\infty)]$, will be less than M_0 , and hence the theorem is proved. ♦ ♦ ♦

6.5 Complexity of the ML_{RP} Automaton

If T_{MLRP} queries are processed before the ML_{RP} algorithm converges, then the time complexity of the ML_{RP} automaton is $O(T_{MLRP})$. If W objects are

moving around in the ML_{RP} automaton, then since each object is allowed to choose any of the W actions, the space complexity is $O(S_{MLRP})$, where $S_{MLRP} = W^2$.

Based on the experimental results we have obtained, we conjecture that

$$O(T_{MLRP}) < O(T_{BAM})$$

$$O(S_{MLRP}) > O(S_{BAM})$$

6.6 Concluding Remarks

In this chapter we have proposed a new variable structure stochastic automaton called the Modified Linear Reward-Penalty (ML_{RP}) automaton to solve the OPP. The updating scheme of this automaton is derived from the well-known Linear Reward-Penalty (L_{RP}). Although the ML_{RP} automaton has been shown to be expedient, a rigorous analysis of its ϵ -optimality is not yet available.

Chapter 7

Conclusions

7.1 Summary of the Results

In this thesis, we have attempted to solve the Object Partitioning Problem (OPP). The intention is to partition a set of objects $\mathcal{A} = \{A_1, \dots, A_W\}$ into R classes, where each class may have a different size, such that frequently accessed objects are placed in the same class. Unlike other methods discussed in the literature which considered the OPP as an optimization problem or as a clustering problem, we consider the OPP from a learning perspective and various learning automata have been proposed to solve it.

To compare our work with work of others, we have studied the solutions proposed by Hammer *et al* and by Yu *et al*. However, since the latter solution is superior to former one, we only compared our solutions with the one by Yu *et al*.

We have first considered the case of partitioning objects into equal-size classes, namely the Equal Partitioning Problem (EPP). The well-known Tsetlin and the Krinsky automata have been suggested for solving this. In this case the partitioning of the objects is done by learning to find an optimal partition from the set of all possible partitions that can be obtained from the given objects. Despite the fact that these two automata have ϵ -optimal properties, they are feasible only when the size of the object set is small. This is because of the exponential growth in the number of partitions as the object size increases.

Besides the above solutions, a new fixed structure stochastic automaton called the Object Migrating Automaton (OMA) has been developed to tackle the EPP. Unlike the Tsetlin and the Krinsky automata which learn to find an

optimal partition from a set of all possible partitions, in this case the abstract objects, which represent the physical objects to be partitioned, are allowed to move around in the OMA. The partition is "learned" according to the action (class) chosen by the abstract objects. In a fairly simple example, this automaton is shown to be expedient. However, experimentally it demonstrates an ϵ -optimal property. Further, these results prove that it converges faster than the Basic Adaptive Method (BAM) proposed by Yu *et al* [27], which is the best known method in the literature for the OPP. This method also demonstrates excellent partitioning capability when the queries are relatively less informative.

In the effort of generalizing the OMA to handle the OPP, we have identified two important properties of the OPP. These two properties show that both the initial arrangement of objects and the movement of objects in an OMA are essential in finding an optimal partitioning of the objects. Using these properties, another new FSSA called the Stochastic Move Automaton (SMA) is developed. This automaton gives the abstract objects the option of stochastically changing actions when a change may be required. The partitioning properties of the SMA have been studied experimentally and its drawbacks for solving the OPP have been analyzed.

From the disadvantages of the SMA, an important property of the OPP has been discovered. This is based on the observation that the transition of objects in the SMA is not ideal in the interior states of an action. This property suggests that the inaccuracy of the SMA is due to an undesirable nature of all FSSA, namely, the fact that it almost always chooses the same action at two consecutive instants even if the action is not desirable. Prompted by this, a new variable structure stochastic automaton (VSSA) called the Modified Linear Reward-Penalty (ML_{RP}) automaton is proposed. This automaton employs an

updating scheme which is a derivative of the well-known Linear Reward-Penalty (L_{RP}) scheme and has been shown to be expedient. In comparison to the BAM, this solution shows remarkable improvements in terms of the rate of convergence.

7.2 Future Research

Although several new solutions to the OPP have been proposed in this thesis, the following problems are still not solved:

- (1) In the case of the Tsetlin and the Krinsky automata, if one can devise a method of computing a partition and its corresponding action from the index of the state of the automaton, then these two automata would be feasible. This observation is true even for large sizes of the object set. However, such a method is still unknown.
- (2) Although the OMA model gives excellent results in solving the EPP, a rigorous analysis regarding its expediency and ϵ -optimality is yet to be found. It is hoped that a solution to this will emerge in the near future.
- (3) Just like the OMA, analyzing the ML_{RP} automaton for its ϵ -optimality is still open. Also, the problem of finding the best parameter values of the scheme is unsolved. Hopefully this will lead to a technique to refine the ML_{RP} automaton to handle the situation when the interacting environment is relatively less informative, namely when P is not close to unity.
- (4) Throughout the thesis we only considered distinct objects. An extension of the OPP is to have replication of objects in various classes. Note that in this case the intersection of the classes produced by the OPP solution would not be null. This variation of the problem is extremely interesting but remains open.

References

1. Arnow, D.M., and Tenenbaum, A. M., "An Investigation of the Move-Ahead-k-Rules", Congressus Numerantium, Proc. of the Thirteenth Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida, Feb. 1982, pp.47-65.
2. Bitner, J.R., "Heuristics that Dynamically Organize Data Structures", SIAM J. Computing, Vol. 8, 1979, pp. 82-110.
3. Gonnet, G.H., Munro, J.I., and Suwanda, H., "Exegesis of Self Organizing Linear Search", SIAM J. Computing, Vol. 10, 1981, pp. 613-637.
4. Hammer, M., and Niamir, B., "A Heuristic Approach to Attribute Partitioning", Proc. of the ACM SIGMOD Conference, 1979, pp.93-101.
5. Hammer, M., and Chan, A., "Index Selection in a Self-adaptive Database Management System", ACM SIGMOD Conference, 1976, pp.1-8.
6. Hendricks, W. J., "An Account of Self-Organizing Systems", SIAM J. Computing, Vol. 5, 1976, pp. 715-723.
7. Kan, Y. C., and Ross, S.M., "Optimal List Order Under Partial Memory Constraints", J. App. Probability, Vol. 17, 1980, pp. 1004-1015.
8. Knuth, D.E., "The Art of Computer Programming, Vol. 3, Sorting and Searching", Addison-Wesley, Reading, MA, 1973, pp.398-399.
9. Lakshmivarahan, S. and Thathachar, M.A.L., "Absolutely Expedient Algorithms for Stochastic Automata", IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-3, 1973, pp.281-286.
10. Lakshmivarahan, S., "Learning Algorithms Theory and Applications", Springer-Verlag, New York, 1981.
11. Lam, K. and Yu, C.T., "An Approximation Algorithm for a File Allocation Problem in a Hierarchical Distributed System", ACM SIGMOD Conference, 1980, pp.125-132.
12. McCabe, J., "On Serial Files with Relocatable Records", Operations Research, Vol. 12, 1965, pp.609-618.

13. Narendra, K.S., and Thathachar, M.A.L., "Learning Automata -- A Survey", IEEE Trans. Systems, Man and Cybernetics, Vol.SMC-4, 1974, pp. 323-334.
14. Narendra, K.S., and Thathachar, M.A.L., "On the Behaviour of a Learning Automaton in a Changing Environment With Routing Applications", IEEE Trans. on Syst. Man and Cybern., Vol.SMC-10, 1980, pp. 262-269.
15. Narendra, K.S., Wright, E. and Mason, L.G., "Application of Learning Automata to Telephone Traffic Routing", IEEE Trans. on Syst. Man and Cybern., Vol.SMC-7, 1977, pp.785-792.
16. Oommen, B.J., and Hansen, E.R., "The Asymptotic Optimality of Two Action Discretized Linear Reward-Inaction Learning Automata", IEEE Trans. on Systems, Man and Cybernetics, May/June 1984, pp. 542-545.
17. Oommen, B.J., "Absorbing and Ergodic Discretized Two Action Learning Automata", To appear in IEEE Trans. on Systems, Man and Cybernetics.
18. Oommen, B.J., "On the Use of Smoothsort and Stochastic Move-to-Front Operations for Optimal List Organization", Proc. of the Twenty-Second Allerton Conference, Oct. 1984, Urbana, Illinois, pp.243-252.
19. Oommen, B.J., and Hansen, E.R., "List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations", Submitted for Publication. Also available as SCS-TR-76, from Carleton University, Ottawa.
20. Rivest, R.L., "On Self Organizing Sequential Search Heuristics", Comm. ACM, Vol. 19, 1976, pp. 63-67.
21. Sleator, D. and Tarjan, R., "Amortized Efficiency of List Update Rules", Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing, April 1984, pp.488-492.
22. Salton, G., "Dynamic Information and Library Processing", Prentice Hall, Englewood Cliffs, New Jersey, 1975.
23. Schkolnick, M., "A Clustering Algorithm for Hierarchical Structures", ACM Trans. on Database Systems, 1977, pp.27-44.
24. Tsetlin, M.L., "On the Behaviour of Finite Automata in Random Media", Avtomat. Telemek. , Vol. 22, 1961, pp.1345-1354.

-
25. Tsetlin, M.L., "Automaton Theory and Modelling of Biological Systems", Academic Press, New York and London, 1973.
 26. Varshavskii, V.I. and Vorontsova, I.P., "On the Behaviour of Stochastic Automata with Variable Structure", Avtomat. Telemekh., Vol. 24, pp. 353-360, March, 1963.
 27. Yu, C.T., Siu, M.K., Lam, K., and Tai, F., "Adaptive Clustering Schemes : General Framework", Proc. of the IEEE COMPSAC Conference, 1981, pp.81-89.
 28. Ullman, J.D., "Principles of Database Systems", Computer Science Press, Maryland, 1982.

END / FIN

2 04 87