National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

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

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

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

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

AVIS

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

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

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

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

Learning Algorithms for Data Retrieval and Storage

by

Radhakrishna S. Valiveti

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Department of Systems and Computer Engineering

Carleton University
Ottawa, Ontario
November 21, 1990

© Copyright
1990, Radhakrishna S. Valiveti

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service   Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

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

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

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

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

ISBN 0-315-68825-4

Canada

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,
Learning Algorithms for Data Retrieval and Storage
submitted by
Radhakrishna S. Valiveti
in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Chairman, Department Of Systems and Computer Engineering

Thesis Supervisor

External Examiner

Carleton University
January 31, 1991

Abstract

Pattern recognition is a fundamental problem in many areas of computer science. It naturally occurs in the process of retrieving data that satisfies certain criteria. The probabilistic approach to retrieval invariably leads to the Bayes' rule for classification, which assumes that the class-conditional densities are available. In the absence of this information, one can either estimate them, or approximate them by using densities that can be easily computed. We first study this problem for the case when the components of the random vectors of interest are either discrete or continuous random variables. We demonstrate how the chi-squared metric can be used to capture the dependence information.

All of data retrieval consists of accessing elements from well-defined data structures. If the form in which the data structure is created initially depends only on the order in which the elements were presented, the expected cost of accessing this structure can be large, but this cost can be minimized by dynamically transforming the structure. The goal of such reorganization is that the system will "learn" the optimal order, or at least be in the optimal state with a high probability. We study two data structures: the singly-linked-list and the doubly-linked list. For the singly-linked-list, we have studied new adaptive heuristics for dealing with the case when sets of elements are requested in a single query. Additionally, we have studied the performance of the classical Move-to-front rule, when used in non-stationary single-element query systems. For doubly-linked-lists, we have compared two existing heuristics and investigated the properties of a new absorbing scheme.

The thesis also considers various adaptive solutions for problems involving string recognition, and keyboard optimization.

Acknowledgements

It has been my great fortune to have known and worked with Prof. B.J. Oomen. His unyielding enthusiasm for research was a constant source of inspiration throughout the duration of this work. I am extremely grateful to him for his excellent guidance, his patience, and the support and warmth that he has extended to me throughout my years as a graduate student.

I would like to thank Profs. B. Pagurek and N.W. Dawes from the Department of Systems and Computer Engineering (Carleton University), Prof. S. Matwin from the Department of Computer Science (University of Ottawa) and Prof. Frank Dehne from the School of Computer Science (Carleton University), for having accepted the responsibility of being on my committee and for providing valuable comments during the thesis proposal. I also wish to thank the External examiner, Prof. S. G. Akl (Queens University, Kingston) for his careful reading of the thesis and for the many suggestions he made during the thesis defense.

I take this opportunity to thank Prof. M.D. Atkinson for initial discussions concerning the computational complexity of the keyboard optimization problem. Thanks are also due to David Ng for some preliminary discussions regarding doubly linked lists. I would like to acknowledge the contribution of Jack Zgierski, who implemented the algorithms contained in the chapters dealing with multiple access queries and keyboard optimization, and also collected the experimental results reported there.

I wish to thank Duncan Chaundy for convincing me that $\text{\LaTeX}$ was the way to go! I would also like to put on record my sincere appreciation for the (unofficial) $\text{\TeX}$ support provided by Rick Mallett of the Computing Services. Peter Chownowski

has also been very helpful in ensuring that the "Sparc" systems in the School of Computer Science supported L^AT_EX even though for quite some time I was the only user. The staff of the School of Computer Science, especially Rosemary Carter, Barbara Coleman and Marlene Wilson deserve special thanks for ensuring that I had an enjoyable and supportive environment.

I am indebted to NSERC and BNR for their financial support during the course of my graduate study. I would like to thank Dr. Jack Dymment and the University Interaction group for their assistance.

Lastly I would like to thank my wife Natana for her love and support during these years of graduate study. Her help in the preparation of this document has been invaluable.

Contents

List Of Tables	x
List Of Figures	xi
List Of Programs	xiii
List Of Acronyms	xiv
1 Introduction	1
1.1 Pattern Recognition	2
1.1.1 Approximating Densities	6
1.1.2 Source Recognition Problem	10
1.2 Adaptive Learning Algorithms	13
1.2.1 Adaptive Data Structures	13
1.2.2 Keyboard Optimization	21
1.3 Organization of the Thesis	23
2 Dependence Trees for Discrete Vectors	26
2.1 Problem Statement	27
2.1.1 Approximation Techniques	27
2.2 Dependence Tree Approximation	29
2.3 The χ^2 statistic	33

2.4	Experimental Results	45
2.5	Summary	53
3	Dependence Trees For Continuous Vectors	54
3.1	Approximating Continuous Distributions	57
3.1.1	Optimum Dependence Tree	58
3.1.2	Optimal Dependence Tree for Normal Distributions	59
3.1.3	χ^2 Metric For Determining Dependence Trees	63
3.2	Estimating Dependence Tree From Samples	69
3.3	Experimental Results	75
3.4	Summary	80
4	Recognizing Sources Of Random Strings	81
4.1	The U-model for Arbitrary Strings	81
4.2	The S-Model for Permutation Generators	87
4.2.1	Notation	88
4.3	Analysis of the S-model	89
4.4	Modeling Distributions Using the S-model	92
4.4.1	Experimental Results	100
4.4.2	Numerical Instability	102
4.5	Use of S-model in Source Recognition	102
4.5.1	Experimental results	105
4.6	Applications of the S-model	107
4.6.1	Ensemble Presortedness Metric	111
4.7	Summary	121

5	Doubly-Linked Lists	123
5.1	Introduction	123
5.1.1	The optimal order in a DLL	124
5.1.2	Scope of this Chapter	124
5.2	Heuristics for DLL	125
5.2.1	Comparison of the SWAP and MTE heuristics	128
5.3	Absorbing schemes for DLLs	133
5.3.1	Absorbing-MTE	134
5.3.2	Implementation Considerations	146
5.4	Summary	152
6	Singly-Linked Lists and Non-Stationary Queries	153
6.1	Overview of Problem	153
6.2	Markovian Switching Query Generators	158
6.2.1	Exact Analysis Using the Composite Chain	159
6.3	Random Query Generator	171
6.3.1	Time Invariant Means	172
6.3.2	Time Varying Means	174
6.4	Experimental Results	177
6.5	Summary	179
7	Multiple Access Queries	181
7.1	Introduction	181
7.1.1	The P-model for MAQs	183
7.1.2	Data Reorganization Under MAQs	184
7.2	Processing MAQs	186
7.3	Avenues for Studying MAQ Heuristics	188
7.3.1	Analysis of the Serialized Query Stream	190
7.4	The Move-to-Front Heuristic for MAQs	202

7.5	The Transposition Heuristic for MAQs	211
7.6	Experimental Results	216
7.7	Summary	222
8	Keyboard Optimization Problem	223
8.1	Problem Statement and Notation	223
8.1.1	Dictionary Representation Mechanisms	226
8.2	Complexity Of the Keyboard Optimization	228
8.3	Previous Solutions to OptKeyboard	233
8.4	Automata Solutions To OptKeyboard	236
8.4.1	Fundamentals of Learning Automata	237
8.4.2	The Krinsky Automaton	239
8.4.3	The Keyboard Learning Automaton	240
8.5	Experimental Results	251
8.6	Summary	257
9	Conclusions	258
	Bibliography	262
A	Proof of Theorem 2.1	272
B	Generating Samples of Normally Distributed Vectors	276
C	Computing Closeness Measure	278

List of Tables

2.1	Accuracy of τ_x . All features are binary valued	48
2.2	Accuracy of τ_x . All features are ternary valued	48
4.1	Keys for RSA scheme	85
4.2	Recognition accuracy for Keys	86
4.3	MLE of ρ vs. Inversions	121
6.1	Average access costs for the MSQG model	178
7.1	MTF_TQS and Linear $\mathcal{P}$	219
7.2	MTF_TQS and Zipf's $\mathcal{P}$	219
7.3	MTF_TQS and Exponential $\mathcal{P}$	220
7.4	TR_TQS and Linear $\mathcal{P}$	220
7.5	TR_TQS and Zipf's $\mathcal{P}$	220
7.6	TR_TQS and Exponential $\mathcal{P}$	221
8.1	Comparison Of 3 Algorithms for Keyboards	253

List of Figures

2.1	Equivalent procedures for finding the maximum likelihood tree from samples.	32
2.2	Rate of Convergence. Discrete Vector case ($N=6$).	50
2.3	Rate of Convergence. Discrete Vector case ($N=10$).	51
3.1	Rate of Convergence. Continuous Vector case ($N=6$).	78
3.2	Rate of Convergence. Continuous Vector case ($N=10$).	79
4.1	Set up for generating data for source recognition. Every source ω_i generates data based on the English bigram distribution. The data generated by the source ω_i is encrypted using the key K_i to yield the output of the pseudo-source ω'_i	84
4.2	Convergence of the Estimate of the S-vector.	103
4.3	Comparison of the Distribution generated by the S-model to the "real" one.	108
4.4	Distributions used by the 4 sources.	109
4.5	Variation in ρ vs Avg. number of Inversions. $M = 16$	119
4.6	Variation in ρ vs Avg. number of Inversions. $M = 24$	120
5.1	Illustration of Absorbing algorithm for DLLs	136
5.2	Absorbing Markov chain used to model Algorithm Absorbing_DLL_1.	140
5.3	A DLL being reorganized.	148

5.4	Illustration of Algorithm Absorbing_DLL_2	151
6.1	The Markov Chain for the list with 3 elements. The MTF heuristic is used to reorganize the list with every access.	155
6.2	Composite Markov Chain	163
7.1	The Markov chain capturing the information about set boundaries in SQS.	192
7.2	Illustrating the MTF_TQS rule	204
7.3	Illustrating the TR_TQS rule	213
8.1	Input graph $G = (V, E)$	231
8.2	KLA with unrestricted characters per key. KLA processes the word $X = x_1x_2 \dots x_N$ and every character x_i of X is rewarded. In the figure, the character x_i is in the action α_k	244
8.3	KLA with unrestricted characters per key. The colliding characters of the word being processed are punished.	245
8.4	KLA with restricted characters per key. The colliding characters of the word being processed are punished.	246
8.5	Rate of Convergence of the three algorithms used for keyboard opti- mization.	254
8.6	Improvement in Collisions with number of models examined. The initial configuration of characters represents the worst case.	255
8.7	This figure illustrates the performance of KLA in the “worst-case” scenario. The initial keyboard and the final keyboard yielding 9 col- lisions have been shown.	256

List of Programs

2.1	Chow's Algorithm to Compute Dependence Tree.	31
3.1	Algorithm to Determine the Optimal Dependence Tree for Normal Distributions.	62
3.2	Algorithm to Determine the Dependence Tree for Normal Distribu- tions as per the chi-squared metric.	67
3.3	Estimate Dependence Tree from Samples.	70
4.1	S-model for permutation generation.	88
5.1	Absorbing scheme for DLLs.	137
5.2	A practical implementation of the Absorbing algorithm for DLLs . . .	150
7.1	The MTF_TQS heuristic for MAQGs	203
7.2	The TR_TQS heuristic for MAQGs	212
8.1	Algorithm Reduce: Reduces SIMP2C to ApproxKeyBoard.	230
8.2	Algorithm KLA.	249
B.1	Algorithm to Generate Samples from a Multi-variate Normal Distri- bution.	277

List of Acronyms

AHS	Adaptive Heap Sort
AI	Artificial Intelligence
DLL	Doubly Linked List
EMIM	Expected Mutual Information Measure
EVOL-M	Evolution Algorithm (Monotonic version)
FSSA	Fixed Structure Stochastic Automaton
IR	Information Retrieval
IS	Insertion Sort
KLA	Keyboard Learning Automaton
KLA-R	KLA with restriction on no. of symbols per key
KLA-U	KLA with unrestricted no. of symbols per key
MAQ	Multiple Access Queries
MAQG	Multiple Access Query Generator
MLE	Maximum Likelihood Estimate
MSQG	Markovian Switching Query Generator
MST	Maximum Spanning Tree
MTE	Move-to-End
MTF	Move-to-Front
NR	Newton-Raphson
PR	Pattern Recognition
RQG	Random Query Generator

RSA	Rivest-Shamir-Adelman
SIMP2C	Simplified 2-Clustering
SLL	Singly Linked List
SQS	Serialized Query Stream
TQS	True Query Stream
TR	Transposition
TSP	Traveling Salesman Problem
VSSA	Variable Structure Stochastic Automaton

Chapter 1

Introduction

Ever since the introduction of the digital computer, unceasing efforts have been directed at expanding the domains to which its computational power can be applied. One of the relatively common applications of the enormous computing power provided by the computer has been in the areas of efficiently storing, retrieving, and manipulating large volumes of data. Thus it is no wonder that it has also been employed to construct large databases, including those that permit the fast and intelligent accesses to the data in question.

Besides the obvious applications, digital computers have also been employed to mimic various facets of human intelligence. The pursuit of this goal forms the key interest of the domain of study termed as Artificial Intelligence (AI). Research in this direction has led to the development of many complex rule-based systems which accomplish extremely intricate tasks which were previously assumed to be requiring human intellect. Indeed, it has been argued that the major focus of this research endeavor is on the deductive processes achieved by the human mind.

Besides studying the human capability to reason with purely deterministic situations, a considerable amount of emphasis has been placed on the study of our capability to deal with uncertainty. Since the ability of living organisms to deal with uncertainty is well-known, engineers have endeavored to incorporate similar

features in various man-made systems. As a result of these efforts, many terms such as pattern recognition (PR), learning, adaptation, etc. were “borrowed” from fields such as psychology, and biology, and each one of them has expanded into a complete avenue of study. Pattern-recognition is simply the ability to recognize patterns and group objects of a similar sort. Learning, on the other hand, refers to the ability to improve the “performance” of the system, based on its past experience.

The topics studied in this thesis fall under two categories. The first is the overall field of pattern recognition and the second consists of problems related to adaptive learning. The intention of this introductory chapter is to give the reader a top-level view of the nature of the problems that have been investigated. Thus, from this chapter, the reader can expect only a cursory, non-specialist view of the various problems tackled and the solutions proposed.

In the first and second sections of this chapter, we shall informally describe the various PR problems studied and these primarily concern the question of obtaining approximations for general probability densities and the problem of recognizing sources of random strings. In the subsequent subsections of this chapter, we shall informally discuss the adaptive learning problems we have researched, and these involve the design and analysis of various adaptive data structures and a study of a new adaptive data compression technique. This introductory chapter concludes with a brief outline of the organization of the thesis.

1.1 Pattern Recognition

One fascinating capability possessed by humans and higher living organisms is their ability to easily perform pattern classification — namely the association of a specific object with one of a set of predefined categories. This ability to perform pattern classification (also referred to as PR), is generally acquired in the following sequence. First of all, a set of indicative features of the objects of interest must be identified. The subsequent step consists of examining the “class of subjects” with a view to extract the chosen features from each member. This information then needs to be

assimilated so as to obtain a condensed representation of the subjects belonging to each of the classes of interest. The final phase consists of using this acquired knowledge to classify the unknown object.

In the above description of a PR system, we have dealt with the problem from an abstract point of view, without referring to a specific application. However, from a more pragmatic perspective, a PR system can be used as a model for the front-end of a data retrieval system. Indeed, PR naturally occurs in situations where, instead of examining new samples, the processing front-end of the data retrieval system must examine a collection of objects and determine their suitability to a specific user query. This situation occurs frequently in Information Retrieval (IR), where, the set of objects is the list of all the documents accessible to readers and the task of the system is to classify each available document as being either *relevant* or *non-relevant* to a particular user's query. Observe that in the case of IR, the feature vectors correspond directly to a vector of binary random variables each of which indicates the presence or absence of a specific index term in the document. This non-trivial application of PR dominates a major segment of our research, and having introduced it in all its brevity, we now return to the generic PR problem.

The primitive step of extracting measurements of features for a specific object is a straightforward one. This process merely relies on transducers to obtain measurements of the salient characteristics of interest. Of course, the difficult problem in this context is one of determining what these salient features are, and invariably a human expert is needed to make this decision. Once the feature extraction has been accomplished, the next step is to gather information on the nature of variation of features within a class. This phase, is customarily referred to as "training" and over the past three decades various avenues for conducting the "training phase" have been investigated. Two well-acclaimed modes of achieving this are the supervised and the unsupervised modes of learning.

Generally speaking, the supervised mode represents a simpler model of learning. In this case, whenever the machine is presented with a sample, it is assisted by an external "teacher" who specifies the class which the sample belongs to. More

specifically, in this training phase, the machine is presented with a sequence of training samples and the information about their respective classes. This aspect is akin to the area of *concept acquisition* in studies concerned with machine learning [63]. In example based inductive learning, the learning system is presented with descriptions of objects from a class. These descriptions usually take the form of a set which specifies attribute-value pairs. Inductive learning can be regarded as a search for general descriptions (also called inductive assertions) which explain the input data that has already been processed by the system, and which can be used to predict the membership of a new object. For any input data, a large number of inductive assertions can be generated with varying degrees of generality. The goal of inductive learning is to gradually refine these assertions as more and more pieces of input data are observed. Inductive learning can be viewed as a search through the description space. This paradigm of learning is very general and is applicable to a large variety of languages capable of symbolically describing the objects in the domain space. Since our focus in this thesis is mainly on statistical PR, we shall not dwell on this aspect any more, but rather refer the interested reader to [63].

In statistical pattern recognition, we take the view that object descriptions are finite dimensional vectors, whose components numerically represent the measurements of interest on the objects being studied. These features are modeled as a random vector $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$, called the feature vector. It is assumed that the vector $\mathbf{X}$ has a distribution that is dependent on the class to which it belongs to. In general, if we denote the set of classes as $\{\omega_1, \omega_2, \dots, \omega_c\}$, the density functions for the feature vector for the class ω_i is typically represented as $p(\mathbf{X}|\omega_i)$. In addition to these densities (referred to as class-conditional densities [18]), it is generally assumed that the system possesses a knowledge of the *a priori* probabilities of the random sample belonging to the underlying c classes. These *a priori* probabilities are typically denoted by the set $\{Pr(\omega_i)|i = 1, 2, \dots, c\}$.

It is well known that the classifier which attains the minimum probability of error is the Bayes' classifier specified by:

$$\text{Classify } \mathbf{X} \text{ as coming from } \omega_i \text{ if : } Pr(\omega_i|\mathbf{X}) > Pr(\omega_j|\mathbf{X}) \quad \forall j \neq i. \quad (1.1)$$

The decision rule specified by (1.1) can be equivalently rewritten as:

Classify $\mathbf{X}$ as coming from ω_i if:

$$Pr(\omega_i)p(\mathbf{X}|\omega_i) > Pr(\omega_j)p(\mathbf{X}|\omega_j) \quad \forall j \neq i.$$

Based on the above reasoning and the vast body of literature supporting Bayesian classification [18, 22], it is clear that if we are to use a classifier based on the statistical properties of the random feature vector, we are immediately faced with the problem of computing the class conditional densities $p(\mathbf{X}|\omega_i)$. As a consequence, when no knowledge about the distribution of the features is available, we cannot attempt Bayesian classification.

Several avenues of estimating the class-conditional densities $p(\mathbf{X}|\omega_i)$ are well known. The simplest of these assumes that although the parametric form of the density function is known, the parameters themselves which characterize the density function are unknown. Thus, for example, if the assumption that the uni-variate random variable is normal (i.e. Gaussian) is justifiable, the parameters that would completely describe the density function are its mean and variance. With a little insight, it can be seen that in this mode, the training phase estimates an unknown *function* by merely estimating its unknown parameters. The various aspects and techniques involved in estimating these unknown parameters from the representative samples have been extensively studied and we refer the reader to the excellent treatises of Fukunaga[22], and Duda & Hart [18].

One of the well-acclaimed rules for estimating the unknown parameters is the Maximum Likelihood Estimation procedure. This procedure selects the unknown parameter values so as to maximize the likelihood of observing the samples used to estimate them [18, 22]. An alternate approach is the well known Bayesian Learning technique of parameter estimation, which regards the parameters themselves as random variables. Indeed, in this case, the uncertainty in the estimated parameter values is represented by imposing a distribution on the possible values that *they* can assume. The representative samples are subsequently utilized to refine or “sharpen” *this* distribution.

For obvious reasons, the above techniques of estimating the density functions are called parametric techniques. The basic premise characterizing these is that the form of the density function which is being estimated is known, and generally the form is assumed to be one of the classical, analytically tractable densities which are unimodal and regular. In real-life situations, whenever the “true” density function being estimated does not closely correspond to one of these, the parametric approach is not applicable. Thus, if the density function is multimodal, alternate strategies have to be explored and this has led to the development of several non-parametric techniques. An example of such an estimation procedure is the Parzen-Window scheme [18, 22]. The latter is an extremely general procedure, and the convergence of the scheme to the arbitrarily complex density function is almost always guaranteed after the scheme has examined a large number of samples. In spite of this advantage, since the amount of information to be stored so as to adequately represent the desired function is enormous, the practical applications of these techniques are limited.

The other interesting approach to estimating density functions is to use approximations involving easily computable density kernels such as the Bahadur-Lazerfeld expansions [18] and the class introduced by Chow and Liu [11]. Since the latter class is extensively studied in this thesis, we shall now explain its salient features and justify its applicability in PR.

1.1.1 Approximating Densities

Consider the case when all the components of the random vector $\mathbf{X}$ are discrete valued. Because of the largeness of the dimensionality of the vector $\mathbf{X}$ (i.e., because of the large magnitude of N), it is both infeasible and impractical to store estimates of the joint mass function $P(\mathbf{X})$ for all possible values of $\mathbf{X}$. It is infeasible because, if each feature x_i could take one of a set of k_i distinct values, the number of estimates that would have to be maintained is of the order of $\prod_{i=1}^N k_i$. Obviously such a scheme would be impractical also. A second alternative is one of approximating $P(\mathbf{X})$ by a well-defined and easily computable density function $P_a(\mathbf{X})$.

When using an approximation density function $P_a(\mathbf{X})$, the question is now one

of measuring how well this function approximates the “real” density function $P(\mathbf{X})$. In order to quantify this “goodness” of approximation, a distance measure between these two density functions must be utilized. One such measure, which is an information theoretic measure, has been known to serve this need. It is given below:

$$I(P, P_a) = \sum_{\mathbf{X}} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} \quad (1.2)$$

It is well known that $I(P, P_a) \geq 0$ and $I(P, P_a) = 0$ only if the approximation is exact (i.e. $P(\mathbf{X}) = P_a(\mathbf{X})$, for all $\mathbf{X}$). Hence $I(P, P_a)$ is a measure of the closeness of the approximation: the smaller this measure, the better the approximation. In other words, the intent in finding the best approximation for a given density function $P(\mathbf{X})$ is to find a density function $P_a(\mathbf{X})$ from the set of available approximations such that $I(P, P_a)$ is minimized.

Since we have earlier argued that it is infeasible and impractical to gather the estimates for the joint density function $P(\mathbf{X})$ for all values of $\mathbf{X}$, an alternate strategy would be one in which we are restricted to collecting information about only the lower-order marginals — which are the joint density functions for a subset of the N features. This leads us to the basic method adopted in all the approximation methods, which can be described as follows: if all marginals of the k^{th} order of $P(\mathbf{X})$ are available, the approximation $P_a(\mathbf{X})$ will be chosen so that its marginals of order k (and less) agree with the corresponding marginals of the underlying density function $P(\mathbf{X})$. Among all such functions which satisfy the constraints imposed by these marginals, the best approximation is the one that minimizes the closeness measure defined in (1.2). It is natural to expect that the quality of the approximation improves as increasingly higher-order marginals are reckoned in the computation.

The first solution to this problem of approximating discrete distributions was presented by Chow *et. al.* in [11]. In this method, the approximation uses only the information gathered about the first and second-order marginals. This constraint naturally leads to an approximation which is based on the *Chain Rule*, with the modification that the conditional probabilities utilized in the approximation are of the type $Pr(x_i|x_j)$. Chow *et. al.* showed that this can conceptually be viewed as a

spanning tree of a complete graph consisting of N nodes in which a node x_i is said to be the parent of x_j , if the probability density function $Pr(x_j | x_i) \neq Pr(x_j)$ and additionally $Pr(x_j | x_i, x_k) = Pr(x_j | x_i)$ for all variables x_k . Throughout this thesis, approximations of this type will be called "tree-type" approximations.

This method due to Chow *et. al.* derives a relationship between the measure of closeness $I(P, P_a)$ defined in (1.2) and the measure of dependence between *all the pairs* of variables. Interestingly enough, this measure is the well known *Expected Mutual Information Measure*(EMIM) ¹ which is defined for all pairs of discrete-valued variables $\{x_i, x_j\}$ as:

$$I^*(x_i, x_j) = \sum_{x_i, x_j} Pr(x_i, x_j) \log \frac{Pr(x_i, x_j)}{Pr(x_i)Pr(x_j)} \quad (1.3)$$

The reader will recognize that this definition for the measure of dependence between variables is very similar to the closeness measure between two density functions (or distributions). In fact, if $Pr(x_i, x_j)$ can be viewed as the underlying distribution, the "approximating" distribution is the one which assumes that the features x_i, x_j are statistically independent. The essential idea in [11] involves computing the EMIM for the $\binom{N}{2}$ pairs of variables, and then selecting the most dominant dependencies. Clearly, this method finds immense application in all problems that include pattern classification as a sub-problem. In particular, the applications to the field of information retrieval can be found in [95] and [96, Chapter 6, pp. 111-143].

It is obvious from the definition of $I^*(x_i, x_j)$ that its evaluation will necessitate evaluation of numerous logarithms. For the discrete valued case, in which each variable x_i, x_j takes on k_i and k_j distinct values respectively, $k_i k_j$ logarithms need to be evaluated. This number gets prohibitive as k_i and k_j increase.

The first problem studied in this thesis is that of investigating the existence and properties of alternate metrics for the computation of the dependence tree. In other words, we shall first attempt to obtain an alternate measure of dependence between two random variables x_i, x_j — one which retains the "power" of the EMIM

---

¹The EMIM metric is closely related to Shannon's entropy function. See [78, pp. 321-322] for more details.

metric defined earlier. Additionally, we seek for a metric which requires a significant reduction in the computational effort involved. In this connection, we would like to note that several ad-hoc metrics have been suggested in the literature but a rigorous study of their effectiveness has been absent. Thus the trees generated by the use of these alternate measures have not been compared to the tree τ^* reported by the EMIM metric.

For this problem, we propose the χ^2 metric as a measure of dependence between pairs of random variables. The problem has been extensively studied in this thesis for the cases in which the components of the random vector in question assume discrete and continuous values. Chapter 2 presents the results for the discrete variable case. In this case, we have proved that the χ^2 metric *locally follows* the EMIM metric, if the features are binary valued, and thus, they increase or decrease together. Additionally, for a restricted family of discrete distributions involving binary valued features, the χ^2 metric has been shown to be equivalent to the EMIM metric, as far as determining the optimal tree is concerned. Our experimental results [91] indicate that the computation of the dependence tree using the χ^2 metric can be accomplished in approximately 20-25% of the CPU time required to compute the best dependence tree as per the EMIM metric.

After studying the discrete variable case, the thesis moves on to consider the case of continuous random variables, and the results obtained are far more significant. Chapter 3 examines the problem of approximating the joint density of the random vector $\mathbf{X}$ when components are continuous and jointly normal. The χ^2 metric, defined in Chapter 2 has been extended in a natural fashion, to capture the degree of dependence between pairs of continuous random variables. It has been shown that for these distributions, the χ^2 metric *always* yields the optimal dependence tree. Apart from this, the estimate of the dependence tree, obtained using the χ^2 metric, based on a finite number of samples, has also been shown to be the Maximum Likelihood Estimate (MLE) of the dependence tree. Chapter 3 discusses the problem in greater detail. These results have been published [92].

Having informally described the problem of approximating discrete and contin-

uous distributions, we shall now discuss another application of PR studied in this thesis.

1.1.2 Source Recognition Problem

Let $\mathcal{A}$ be a finite alphabet. We have c sources belonging to the set $\{\omega_1, \omega_2, \dots, \omega_c\}$, which generate strings composed of the symbols of the alphabet $\mathcal{A}$. Let us assume, for simplicity that these sources generate strings having a fixed length M , which in practice, can be the buffer size. This means that each source generates one of a set of $|\mathcal{A}|^M$ possible strings, at any given time. Additionally each source generates a string from this set according to its *own* distribution. The problem for a receiver is to identify the (possible) source, from a sequence of random strings. This problem, referred to as the Source Recognition Problem, has been studied in the framework of supervised pattern recognition.

As alluded to earlier, in supervised pattern recognition, the system undergoes a period of training before it can be used for classification. In such a pattern recognition system, in the training phase, the input to the system is a set of training samples. From these samples, features are extracted with the understanding that the information contained in the *features* essentially constitutes the information in the samples. Once the features have been extracted, typically, the stochastic properties of the features are estimated, and these are subsequently used in the testing phase. When an unknown sample is received, the same features are extracted from the sample and using a pattern classification strategy, the sample is assigned to one of c distinct classes.

The technique which we use in the Source Recognition Problem is analogous in principle. However the technique is distinct because even in the training phase, each input is not a single sample but a finite stream of random strings, from a particular source. Since the information presented is essentially symbolic in nature, a syntactic "front-end" sub-system must be used before a pattern recognition strategy can come into effect.

A very obvious approach to attack the above problem exists. Given that “only” $|\mathcal{A}|^M$ strings could be generated by the sources, the brute force approach would suggest that the features to be extracted consist of the $|\mathcal{A}|^M$ estimates per source — with each estimate denoting the probability of occurrence of a specific string. This approach is impractical even if $|\mathcal{A}|$ is as small as 2. The other alternative to feature extraction is to construct a model of the string generation strategy used by the source and then to estimate the parameters of the model. One simple minded model assumes that the source ω_i generates each symbol of the output string *independently*, according to a fixed distribution U_i . This distribution is completely specified by a vector of dimension $|\mathcal{A}|$, typically called the Unigram distribution. More sophisticated models would account for statistical dependence between adjacent symbols. When the dependence is merely specified in terms of the immediately preceding symbol, we have the familiar bigram model of string generation. Higher order dependencies can easily be considered. Apart from its application in source recognition, some other applications of such a modeling strategy are described in Chapter 4.

To explore the idea of statistical dependence further, let us consider an extreme form of dependence analogous to the “without replacement” model of experimentation. In this mode, once a symbol has appeared in the string, it cannot appear again in the string. If we assume that $|\mathcal{A}| > M$, then the sources are generating random subsets of size M from the alphabet $\mathcal{A}$. On the other hand, if $|\mathcal{A}| = M$, the sources are generating permutations of the alphabet $\mathcal{A}$. Recognizing the sources in the case of such dependencies will also be studied in this chapter.

In order to extract features from a stream of permutations, a very simple method would recommend maintaining the $M!$ frequency counters which estimate the probabilities with which the given source generates the various permutations. Clearly such a strategy is infeasible even when M is as small as 10. Considering the above, our first major deviation is to model each permutation generator in a fashion which requires a linear (as opposed to $M!$) number of parameters. We refer to this model as the S-model, and it is not out of place to mention that these parameters constitute

a probability distribution in themselves.

In the training phase, when we are given a stream of permutations coming from a given source, we estimate the parameters of the S-model of this source. These parameters will be referred to as the S-vector or the S-parameters. It will be seen that each stream of permutations yields us a single S-vector, and if we are given a training set consisting of c such independent streams (i.e. one from each source), we can estimate the S-vector for each source. In other words, at the end of the training phase, the system has c estimates for the S-model parameters of these c sources. We refer to these parameters as $\hat{S}_1, \hat{S}_2, \hat{S}_3, \dots, \hat{S}_c$, and the techniques involved in computing these estimates will be rigorously discussed in detail in this chapter.

Coming now to the testing phase, the problem is essentially one in which we associate a stream of permutations with an unknown source, with the intent of minimizing the probability of misclassification. Given this stream, say Q_{test} , the strategy which could be proposed is as follows. By examining Q_{test} , the Maximum Likelihood Estimate of the S-parameters which could have generated it can be computed. Let $\hat{S}_{test}$ be the vector obtained by this process. The classification can be achieved by comparing $\hat{S}_{test}$ to each one of $\hat{S}_1, \hat{S}_2, \hat{S}_3, \dots, \hat{S}_c$, and the stream Q_{test} can be associated with the source whose S-parameters are closest to $\hat{S}_{test}$. In this context, one possible measure of closeness is the Euclidean Metric.

Although the above method conforms to that used in traditional pattern recognition systems, we have opted to choose a different route. Viewed from our perspective, the above approach can be quite erroneous because in the testing of one stream of permutations, Q_{test} , the number of permutations presented to the system, (i.e. the size of Q_{test}) could be very small. Thus even though accurate estimates of the S-model parameters of the sources can be obtained in the training phase, the accurate estimation of $\hat{S}_{test}$ may not be possible. If the number of permutations in Q_{test} is small, the estimate $\hat{S}_{test}$ is either not obtainable or is grossly inaccurate, because of the singularities that are encountered in the estimation process. Thus such a minimum distance classifier cannot be expected to perform in any reasonably accurate fashion.

As opposed to this, our strategy utilizes the maximum likelihood classifier. Rather than resorting to a parameter estimation procedure in the testing phase, in our strategy, we compute the probability with which a particular source ω_i could have generated Q_{test} . The source ω_j for which this probability is maximized is declared to be the generator of Q_{test} . This computation encounters no singularities and thus the classification achieved is both accurate and fast, even when the size of Q_{test} is small.

The data that we analyze is symbolic and thus, it is clear that a syntactic “front-end” is mandatory. Furthermore, the information contained is not merely in the strings but in the arrangement of the symbols within a string. Thus the structure of the stream of random strings itself contains information of paramount importance. Finally once this piece of information has been gleaned, statistical pattern recognition must come into effect during the estimation and classification phases. Thus, it is our belief that our method [93] presents a unique blend of syntactic, and statistical pattern recognition.

1.2 Adaptive Learning Algorithms

1.2.1 Adaptive Data Structures

In all cases of data retrieval, the data to be stored is maintained in a data structure which is appropriate for the application. Many applications make use of a data structure which is created during the initialization phases of a module, and is thereafter manipulated. In all these “pre-created data structures”, the form of the structure depends solely on the order in which the elements are inserted into it. This method of creating the data structure has the obvious flaw that it does not place the elements in the structure so as to maximize the data retrieval rate. For example, in a linear list, in order to minimize the average access times, we would like to locate the more frequently accessed elements near the front of the list. Although this is an intuitive observation, it can be proved formally that the optimal order indeed corresponds to one in which the elements are ordered in the decreasing order

of access probabilities. Similarly, in the case of a binary search tree, it would be generally desirable that the frequently accessed elements are close to the root of the tree.

Clearly in all the above examples and all the well-known data structures, a prior knowledge of the access probabilities of the elements can always be used so as to arrive at the optimal form of the structure — one which yields the lowest expected cost. The module responsible for creating and accessing the data structure would use the information about the element access probabilities and evaluate the optimal structure. Query processing is subsequently achieved by referring to this optimal data structure.

In the absence of the information about the distribution of accesses to the elements, one has the obvious alternative of first estimating the access probabilities. Subsequently, after a sufficiently large number of accesses have been made, (and the estimates are “reasonably” close to the true values,) the optimal structure corresponding to this distribution can be determined and used in all subsequent accesses. The approach is valid, and possibly acceptable, if the access distribution is stationary, i.e., time invariant. The method immediately fails if the access distribution changes over a period of time with the effect that the initially chosen structure is no longer cost effective. Furthermore, the overall philosophy of using additional memory to maintain the statistics is generally discouraged in the literature.

At this juncture, we would like to point out that there do exist methods that do not rely on knowledge of either the true access probabilities or their estimates. To motivate these methods, let us consider the access costs in traditional data structures. The worst case access costs for a linear list and a balanced binary tree are $O(N)$ and $O(\log N)$ where N is the number of elements in the structure. Clearly the access cost grows with the number of elements in the structure. Ideally we would like have a $O(1)$ cost for each element of the structure. Hashing based schemes aim to attain this goal. The basic idea in hashing based methods is to use a “hash function” to compute an address based on the key being retrieved, which is subsequently used to retrieve the relevant records. Observe that if this mapping

from the key space to the hash function space can be rendered unique, the desired goal will have been achieved. Since the key space is typically very large, this is not a viable approach. Thus if two distinct keys map to the same hash function value, a collision is said to have occurred. In this case these schemes rely on maintaining “overflow” buckets to maintain the records that collide. As the number of collisions increase, these schemes quickly deviate from the $O(1)$ goal they were designed to attain.

Many authors have proposed solutions to the problem of collisions in hashing schemes. As a consequence, schemes such as dynamic hashing, extendible hashing, and virtual hashing have been proposed [19, 20, 86, 54, 42, 49]. Although these schemes are distinct in their own rights, viewed from our perspective (of adaptive data restructuring using heuristics) these schemes are similar in spirit. For the sake of brevity, we shall not survey this area but mention the key results reported in a scheme due to Fagin *et. al.* which is probably representative of the entire family [20]. In [20], Fagin *et. al.* propose a scheme called the “Extendible Hashing Technique”, in which the hash function adapts as the structure undergoes modifications because of insertions, deletions etc. Thus their scheme completely avoids the problem of overflows mentioned earlier. The results reported by Fagin *et. al.* are very powerful. In [20], the authors prove that in the worst case, the total number of “page” accesses to retrieve any element of the structure is $O(1)$. In [86] Tamminen proposes a variant of the extendible hashing scheme which incorporates overflows and yet guarantees the $O(1)$ worst case access time. It must be noted however that this performance is attained by maintaining an auxiliary structure, the directory, in order to obtain the desired access times. In this thesis, we are interested in exploring methods that do not guarantee $O(1)$ access times, but simultaneously do not require any elaborate support structures. Thus we shall investigate the improvements that can be attained by reorganizing the data structure, without relying on estimates of access probabilities.

Many heuristics which gradually transform a data structure with queries have appeared in the literature [7, 26, 33, 81]. Among the thoroughly investigated structures

are the Linear list or Singly-linked-list (SLL) and Binary Search Trees. Structure transforming heuristics are merely rules for altering the structure as the elements are accessed. Two types of heuristics have been studied in the literature which are the ergodic and absorbing heuristics. Generally speaking, in the case of the ergodic heuristics, the structure is modified in response to every element access. These transformation rules are often sensitive to the position at which the requested element was located in the structure. In terms of the asymptotic properties of such rules, the structure can eventually be in any one of its possible configurations. These schemes, are therefore modeled using ergodic markov chains [37, 40].

For concreteness, consider a SLL, which is always searched, beginning at the left end. Some examples of ergodic heuristics for the SLL are:

- (i) Move-to-Front (MTF): In this case the accessed element is moved to the front of the list.
- (ii) Transposition (TR): Where the accessed element is moved towards the front of the list by swapping it with the preceding element.
- (iii) Move- k -ahead: In this rule, the accessed element is moved k positions toward the front of the list, unless it is found in the first k elements, in which case it is moved directly to the front of the list.
- (iv) POS- k : In this heuristic the accessed element is moved to the k^{th} position if it appears after this position in the list. If the element appears in positions 2 through k , it is transposed with the directly preceding element.

In all these cases, the element is not moved if it appears in the first position. Detailed analyses of these schemes can be found in [7, 9, 26, 30, 32, 81]. An excellent survey of SLL reorganizing heuristics appears in [33]. These rules will be referred to in Chapters 6, 5 and 7.

As opposed to the ergodic schemes briefly alluded to earlier, the absorbing schemes [72, 73, 74] are characterized by the fact that they are designed with a view to alter the configuration as long as some chosen criterion is not satisfied.

Once this criterion is satisfied (e.g. “every element is moved once”, or “‘ M ’ moves have been made”) no further reorganization takes place and the structure is said to have “converged” to this configuration. Most of the absorbing schemes utilize a parameter which controls the accuracy of the scheme. By a suitable choice of the parameter (or as a limiting case when the parameter value approaches infinity) the probability of the scheme converging to the optimal order can be made as close to unity as desired. Absorbing heuristics are clearly to be preferred, if the access distribution is known to be time invariant or “stationary”.

The research reported in this thesis in the field of SLLs deals primarily with non-traditional query mechanisms and will be stated presently. But to bring the results in the right perspective we deviate marginally and briefly discuss the work done in the area of adaptive Doubly-Linked Lists (DLLs) and mention the heuristics used to transform them for the traditional query system.

1.2.1.1 Adaptive Doubly-Linked Lists

Although SLLs have been very extensively studied and analyzed, very little work has been done to study the adaptive restructuring of Doubly-Linked lists (DLLs). To the best of our knowledge, the only papers in this regard are [60, 69]. The DLL can be defined as a linear list in which each item in the linked list has pointers to both its successor and its predecessor [43]. This structure facilitates searching the list commencing from either end. Various applications of DLLs are catalogued in [88].

Formally speaking, a DLL, D , consists of the set of N records $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ stored in some order. Accesses to the record R_i are made from the left and right with probabilities s_{iL} and s_{iR} respectively. These probabilities obey the constraint that:

$$\sum_{i=1}^N (s_{iL} + s_{iR}) = 1.$$

We assume that the record accesses are statistically independent and identically distributed. We also assume that the starting position of the search for a record is

independent of the record itself. In our case, we will always commence the search for an element from the end at which the element was requested.

It can be shown that the order which yields the minimum average retrieval cost is one which arranges the records in the descending order of the quantity $s_{iL} - s_{iR}$. We shall first concentrate on the behavior of two ergodic heuristics to reorganize DLLs — which are the Move-to-End (MTE) and SWAP heuristics. The MTE heuristic, proposed by Matthews *et. al.* is obtained by extending the classical MTF for SLLs as follows [60].

For ease of explanation, we denote an arbitrary configuration of the DLL D by the permutation $\langle R_{i_1}, R_{i_2}, \dots, R_{i_N} \rangle$ where $\langle i_1, i_2, \dots, i_N \rangle$ is a permutation of the integers $1 \dots N$. The MTE operates as follows. If the element R_{i_j} is accessed from the left and it is not the first element (i.e., $j \neq 1$), the accessed element is moved to the left end of the list. After this movement operation the new configuration is $\langle R_{i_j}, R_{i_1}, R_{i_2}, \dots, R_{i_{j-1}}, R_{i_{j+1}}, \dots, R_{i_N} \rangle$. On the other hand, if the element R_{i_j} is accessed from the right (and $j \neq N$, i.e. the element is not already at the end), the accessed element is moved to the right end, yielding the list: $\langle R_{i_1}, R_{i_2}, \dots, R_{i_{j-1}}, R_{i_{j+1}}, \dots, R_{i_N}, R_{i_j} \rangle$.

The SWAP heuristic was proposed by Ng and Oommen [69], which can be described as follows. As before, let an arbitrary configuration of the DLL be $\langle R_{i_1}, R_{i_2}, \dots, R_{i_N} \rangle$. In the SWAP heuristic, whenever a record R_{i_j} is accessed from the left, the list is reorganized by swapping R_{i_j} with $R_{i_{j-1}}$. We note that no swap is performed if $j = 1$ and the element being accessed is at the left end of the list D . On the other hand, if the search for the element R_{i_j} commenced at the right end, we swap R_{i_j} with $R_{i_{j+1}}$, unless $j = N$. The special case identified corresponds to the case when the R_{i_j} happens to be the last element in the list.

Rivest showed that the TR heuristic always has a lower expected cost when compared to the MTF heuristic[81]. Intuitively there is good reason to believe that SWAP should perform better than the MTE rule. While we conjecture that this is indeed true, a rigorous proof is by no means simple. In Chapter 5 we establish sufficient conditions which will ensure that SWAP is better than MTE. Apart from

this, we also propose an absorbing heuristic for DLL, and derive the necessary and sufficient conditions to be satisfied by the family of distributions for which this scheme converges to the optimal order with a probability as close to unity as desired.

1.2.1.2 SLLs and Non-stationary Query Distributions

Having examined DLL heuristics, the thesis examines SLL heuristics under different conditions of query generation. The assumption made in all studies of adaptive SLLs has been that the access probability vector is time invariant, and that individual accesses are statistically independent. In Chapter 6 we study the impact of relaxing the assumption that the access distribution is stationary. In this chapter, we propose various models of non-stationary query generation. We first propose a model called the Markovian Switching Query Generator (MSQG) in which the query generator is *itself* modeled as a finite Markov Chain. In each state of this chain, the access probability vector is an arbitrary unknown probability vector. Our model is a natural generalization of the model assumed by Testlin [89] and Lam *et. al.* [48]. As a result of this model, we introduce the notion of “locality” in the sequence of elements accessed. Our aim is to study the performance of well-known heuristics such as MTF and TR, when dealing with such a dependent stream of queries.

The question of analyzing the behavior of SLL heuristics, interacting with a MSQG is far from trivial. To convince the reader of the complexity involved, we first examine the classical MTF heuristic when the access distribution is stationary. Let us assume that the SLL consists of the elements of $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ which are accessed according to the distribution $[s_1, s_2, \dots, s_N]^T$. In the Markov chain representing the MTF heuristic, the access probabilities s_i 's appear as transition probability. If the s_i 's are time-invariant, the chain is homogeneous and its convergence is assured. In the case of the MSQG, the s_i 's become functions of time n and the resulting chain becomes non-homogeneous. A rigorous analysis of this system requires that we construct a composite Markov chain, representing the states of the query generator and the list. Considering the large number of states in the composite chain, this is not a viable analytical approach. In Chapter 6 of this thesis, we

attempt a transient analysis of a “lumped” version of this composite chain.

Having studied the MSQG model, we propose a far more general model of query generation called the Random Query Generator (RQG). Under the RQG model, the access probability vector at each time instant is a random vector. The MTF and TR heuristics have been analyzed under this model of query generation and the results of the analysis are also found in Chapter 6.

1.2.1.3 Multiple-Access Queries in SLLs

The thesis then progresses to relax the key assumption made in all the research in SLLs to date. The implicit assumption is that the queries have been composed of accesses to **single** elements of the structure. In Chapter 7 of this thesis we shall investigate the effects of relaxing this assumption and study the implications of a generalized query system. We study this generalization only for the case of the SLL, which is the list consisting of the elements $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$. Coming now to the new types of queries that are permitted, in this study, we shall allow queries to consist of accesses to any subset of the elements of $\mathcal{R}$. These generalized queries are termed “Multiple Access Queries” (MAQ) and the underlying query generation mechanism will be referred to as the Multiple Access Query Generator (MAQG).

A straightforward method to model MAQG becomes obvious when we see that such a generator is fully described by specifying the probabilities of accessing all the 2^N possible subsets of $\mathcal{R}$. This being impractical, we propose an (approximate) query generation model, based on exactly N parameters. For concreteness, we denote this parameter vector as $\mathcal{P} = [p_1, p_2, \dots, p_N]^T$. Under the MAQG, queries can include more than one element of the list. Moreover, each record R_i is included in a specific query, independent of others, with a probability p_i . Under this model of query generation, we show in Chapter 7 that the optimal list order which minimizes the expected retrieval cost is one which arranges the list elements in the descending order of p_i 's. The performance of generalizations of the traditional heuristics such as MTF and TR have been analyzed and the results are presented in Chapter 7.

The final adaptive learning problem studied in this thesis involves a novel application of learning automaton in the area of data compression. This problem is informally described below.

1.2.2 Keyboard Optimization

All of text processing involves manipulating the symbols of an alphabet and in almost all cases this alphabet is finite. For example, the most restricted alphabet is the binary set 0,1, and the alphabet encountered for English text is the set of 26 characters a...z. To distinguish between the words of a language, customarily, various punctuation marks have been defined, the most common one being the delimiter "space" (represented throughout this paper with the symbol '_'). Of course, other punctuation marks such as the comma, semicolon, etc. are not only needed to distinguish the boundaries of the words but also to impart a semantic implication to the text.

Most of text processing functions that are available have a complexity which increases with the size of the alphabet. For example to evaluate the distance between two words, the optimal algorithm requires quadratic time [1, 97, 101] if the size of the alphabet is infinite. Indeed, this is a lower bound on the complexity of the computation. This will also be the lower bound on the complexity of the computation of other comparison metrics involving strings such as the longest common subsequence and the shortest common supersequence, etc. [1, 34, 35, 55, 57, 101]. However, the complexity of this computation decreases as the size of the alphabet decreases and thus it is often advantageous to work with alphabets of small size [1, 59, 101].

Apart from considering the infinite alphabet case, there are many cases where decreasing the size of the alphabet is both advantageous and convenient. For example, consider the Touch-Tone telephone keypad. Associated with every digit in the set $\{2 \dots 9\}$ is a set of characters listed as C_i below :

$$\begin{aligned} C_1 &= \{\} & ; & C_2 = \{abc\} & ; & C_3 = \{def\} \\ C_4 &= \{ghi\} & ; & C_5 = \{jkl\} & ; & C_6 = \{mno\} \\ C_7 &= \{pqrs\} & ; & C_8 = \{tuv\} & ; & C_9 = \{wxyz\} \end{aligned}$$

Let us assume that the digit '0' is used to represent the delimiter, '.'. Observe now that instead of transmitting English text using the English alphabet one can just as easily transmit the text using the symbols 2...9. Thus, instead of transmitting the string 'box' we could just as easily have transmitted the digit string '269'.

There is one fundamental disadvantage to this approach, and that is that the representation of the words can now be ambiguous. Thus, using the above example, if the Touch-Tone keypad was used for transmission, observe that the string '269' could represent both the English words 'box' and 'boy'. In this section, we consider the problem of assigning the letters of the alphabet to the various digits so as to minimize the ambiguities.

Not only is such an ambiguity minimizing keyboard assignment technique advantageous — it is also mandatory in many cases. People who are physically handicapped [52, 53, 64, 65] and constrained to use a limited keyboard have to be able to transmit all the characters on their constrained keyboard. Often, not only is the keyboard limited in size, but the handicapped user is also constrained in terms of his or her vertical and horizontal motion capabilities [52, 53]. It would, of course, be tremendously powerful if the handicapped user could use a modified keyboard and be guaranteed to communicate unambiguously.

Various other applications of such a keyboard assignment technique come to mind. If the dictionary was the set of all the names in a telephone directory, and if the directory was represented in accordance with the keyboard assignment strategy, a telephone company could easily extend the dial-by-number facility currently provided to permit the user a dial-by-name option. Obviously, this would save the user the tedious task of remembering telephone numbers. The user could dial a person of his/her choice provided the keyboard assignment had minimum ambiguity.

Indeed, in a more general setting, this technique can be used to design keyboards for any language in which the application uses a single key symbol to represent a set of characters. A powerful consequence of such a mapping technique is the resulting increase in the channel utilization, as is explained below. Consider the case when the alphabet in question is the English alphabet augmented with the

delimiter '.'. To transmit 27 distinct characters a minimum of five bits must be assigned for each character. However, if all the characters are assigned to seven digits and the delimiter, '.', is uniquely assigned, then a maximum of three bits is sufficient. The data compression is significant – of the order of 40%. Obviously, in this case too, the string need not be uniquely transmitted. However, if the keys are assigned appropriately, ambiguities can be correspondingly minimized, and in the case of ambiguous transmissions, a simple (possibly, user friendly menu-driven) selection technique can be used by the transmitter to exactly communicate which of the ambiguous words have to be transmitted. Thus, if the string "269" represented the English strings "box" and "boy", by appropriately prompting the user, the transmission can be disambiguated by allowing "269*1" to represent "box" and "269*2" to represent "boy".

In Chapter 8 we prove that the problem as stated above is NP-hard. This naturally leads us to seek fast and approximate solutions to the optimization problem on hand. After briefly mentioning the only reported solution to the problem, we report a fast learning automaton-based solution to this problem. Chapter 8 also contains experimental results demonstrating the power of this solution. These results have been published[77].

1.3 Organization of the Thesis

The organization of this document is as follows.

Chapter 2 investigates the dependence tree based approximation due to Chow *et al.* for discrete random variables. We demonstrate that the best dependence tree can be obtained if the EMIM metric for all component pairs is evaluated. Subsequently, we investigate the possibility of alternate metrics which are easily computable, but which retain the capability to yield the optimal tree. We present the χ^2 metric and derive its properties. Experimental results demonstrating its speed of convergence and accuracy are also presented.

In Chapter 3 we consider the problem of approximating the distribution of a con-

tinuous random vector $\mathbf{X}$. All the results with regard to this problem are presented here and in particular we show that for normally distributed vectors, the χ^2 metric always yields the optimal tree.

In Chapter 4 we study the problem of distinguishing between several sources of random strings. We first consider the problem of distinguishing between sources which generate arbitrary strings and then proceed to analyze the estimation and classification aspects of recognizing sources generating permutations. We present an application of the former model in a scenario involving encrypted messages. One application of the latter model which we have studied in detail in Section 4.6.1 is that of arriving at a single-parameter characterization of an ensemble of data sequences.

The use of learning in adaptive data retrieval and compression is subsequently tackled. First of all, Chapter 5 begins by comparing two existing ergodic heuristics to reorganize DLLs. The two heuristics which we have compared are the MTE and SWAP. Additionally we present a new absorbing heuristic for DLLs and derive the necessary and sufficient conditions to be satisfied by distributions in order that the scheme converge to the optimal order with a probability as close to unity as desired.

Having studied DLLs under a traditional model of query generation, in Chapter 6 we study the well known heuristic MTF under a new query generation model, in which the access distribution may be time variant. We propose two models of random query generation. In the first model, we model the query generator *itself* as a Markov chain. In this model, the access probability vector being used by the query generator is a function of the state. In the second model, we permit the access probability vector to be *random* vector. We investigate the performance of the MTF heuristic for SLLs, under the new models of query generation.

Chapter 7 studies the effect of permitting the queries to access any subset of the contents of a linear list. The well-known heuristics MTF and TR are extended so as to generate new heuristics capable of operating under MAQG. These generalizations of MTF and TR are called MTF.TQS and TR.TQS respectively. The MTF.TQS heuristic has been analyzed under this model of query generation. A comparison of these heuristics with their traditional counterparts has also been achieved.

Chapter 8 considers the problem of keyboard optimization. We first demonstrate that the problem is NP-hard. Subsequently, we present the only reported solution and then present an adaptive learning automaton solution. The proposed algorithm is experimentally shown to be superior to the reported evolutionary (genetic) algorithm.

Each of the Chapters 2- 8 conclude with a summary of the results presented in the body of the chapter. Chapter 9 concludes the thesis and briefly discusses possible avenues and directions for future research.

Throughout this thesis, we shall use “{ }” to represent sets, and “[ ]” to represent vectors. Tuples, permutations and ordered sequences will be represented using “⟨ ⟩”. Furthermore, in order to refresh the memory of the reader, we mention the meanings of the acronyms in the respective chapters, even though the introductory chapter may have introduced the acronym. Besides these, a separate list of acronyms has also been provided.

Chapter 2

Determining Dependence Trees for Discrete Random Vectors

Pattern recognition naturally arises in several real world problems such as communication systems, information-retrieval systems, medical diagnosis, and scene analysis. It was already alluded to in the introductory chapter that the Bayesian classifier can be obtained only if we have an accurate knowledge of the densities $P(\mathbf{X}|\omega_i)$, referred to as the class-conditional densities. Of course, if all the features are discrete valued, we would be dealing with discrete probabilities (Dirac Delta Functions) instead of probability densities. The information about the class-conditional distributions is typically obtained by a process referred to as "training". In this phase, the pattern classifier assumes the model for the distribution of features and uses the known samples to estimate the parameters that characterize the distribution.

Owing to the fact that the number of samples is small, and that real machines often impose limitations on available storage, it is often necessary to restrict the amount of information that can be collected from the samples. To simplify the process of training, various simplifying assumptions can be made. The most common assumption is that the individual features of the feature vector are statistically independent. While this assumption simplifies the mathematical formulation, it does not adequately model the real-life situation well. In this chapter, we study the problem of representing the information regarding the stochastic dependence between the features. In Section 2.1, we shall consider the case when the feature

measurements are discrete-valued. Clearly, this constitutes a large number of cases in the field of traditional pattern recognition, and indeed includes the entire field of problems encountered in information-retrieval [95, 96].

2.1 Problem Statement

The central issue in tackling this problem is one of approximating the joint distribution/density $P(\mathbf{X})$, where $\mathbf{X}$ is the vector $[x_1, x_2, \dots, x_N]^T$. Because of the largeness of the dimensionality of the vector $\mathbf{X}$, it is both infeasible and impractical to store estimates of the joint density function $P(\mathbf{X})$ for all possible values of $\mathbf{X}$. It is infeasible because, if each feature x_i could take one of a set of k distinct values, the number of estimates that would have to be maintained is of the order of k^N . Obviously such a scheme would be impractical also. A second alternative is one of approximating $P(\mathbf{X})$ by a well defined and easily computable density function $P_a(\mathbf{X})$.

2.1.1 Approximation Techniques

In the introductory chapter, we had presented a cursory view of the techniques used to approximate distributions. The key practical constraint is that the joint density function $P(\mathbf{X})$ cannot be estimated for all possible values of its argument. Additionally we must merely rely on estimates for the lower-order marginals. The method introduced by Chow *et. al.* [11] utilizes only the information about the first and second order marginals and permits only tree-type dependencies in conjunction with the information theoretic metric referred to as I^* .

In the second type of approximations proposed by Ku *et. al.* in [46], the dependence between the components of the feature vector are not restricted to a tree-type dependence model. Once this constraint is relaxed, a very general method results and the paper [46] describes an extremely interesting and fascinating iterative method to obtain the best approximation possible, with the available information. Since this problem is not the primary concern of our current study, the details of the iterative procedure are not included here.

To view our results in the right perspective, we shall first describe the method presented in [11]. The current study focuses on the use of an alternative new measure, I_X , used to quantify the dependence between variables. This measure, I_X , is obtained by computing the χ^2 distance between pairs of random variables and is much more easily computable when compared to the Expected Mutual Information Metric (EMIM), I^* , defined in (1.3), and used in [11]. Section 2.3 defines the new metric I_X and derives some of its properties. We note in passing that it was proved in [100] that the EMIM metric introduced in (1.3) possesses the chi-square property asymptotically. We, however, prove much more powerful relationships between the two metrics I_X and I^* even for non-asymptotic conditions, and the consequences are immediately obvious. Furthermore, for a restricted class of probability distributions, the metrics I_X and I^* are shown to be equivalent and thus optimal as far as the problem of finding the dependence tree is concerned. Although the new metric I_X is non-optimal in the most general case, experimental results clearly demonstrate its excellent accuracy.

Our experimental results point out that in cases when the underlying distribution is not based on a tree, the new metric generates a tree that is only marginally non-optimal. In the case when the dependence between variables is indeed of the tree type, in addition to the theoretical properties proved, the convergence to the "real" tree is observed in all experiments conducted. Experimental studies have provided evidence that if $\mathbf{X}$ is a random binary vector, the computation of the best tree based on the I_X metric can be usually achieved in about 20-25% of the time required to perform the same operation if the I^* metric is used. The savings in computation time will be even more pronounced if we are working with multi-valued discrete random variables. In other words, the computation of the dependence tree using I_X is almost an order of magnitude faster than the corresponding computation based on the I^* metric. The experimental results are presented in Section 2.4.

2.2 Dependence Tree Approximation

To introduce the method due to Chow *et. al.* [11], we use the following well established rule, *the Chain Rule*, which expresses the joint probability distribution in terms of conditional probabilities:

$$P(X) = Pr(x_1)Pr(x_2 | x_1)Pr(x_3 | x_1, x_2) \cdots Pr(x_N | x_1 \dots x_{N-1}) \quad (2.1)$$

We notice that in this expression, each variable is conditioned on an increasing number of preceding variables. In this context, it is important to point out that estimating the k^{th} term of this equation (i.e. a conditional probability term in which a variable is conditioned on $k - 1$ other variables), requires maintaining the estimates of all the k^{th} order marginals. If we restrict ourselves to computing only the (first and) second-order marginals, we are assured that we can compute all the conditional probabilities of the form $Pr(x_i | x_j)$. Thus we explore the approximation that results if we ignore the conditioning on multiple variables, implying that we shall attempt to retain only dependencies on at most one variable. This leads us to the following approximation:

$$P_a(X) = \prod_{i=1}^N Pr(x_i | x_{j(i)}) \quad \text{where} \quad 0 \leq j(i) < i. \quad (2.2)$$

Notice that in (2.2), each variable is dependent on exactly one variable that has appeared earlier in the equation. To include the case when $j(i) = 0$, we use the convention that x_0 is a dummy variable, which does not influence any other variable. With this understanding, $Pr(x_i | x_0)$ will be the same as $Pr(x_i)$.

In the above case, there is a “natural ordering” among the components $\{x_i | i = 1, 2, \dots, N\}$ of the vector $\mathbf{X}$, such that the variable x_i was conditioned on $x_{j(i)}$, where $j(i) < i$. In the general case, the product approximation takes the form:

$$P_a(X) = \prod_{i=1}^N Pr(x_{m_i} | x_{m_{j(i)}}) \quad (2.3)$$

where, $\langle m_1, m_2, \dots, m_N \rangle$ is a permutation of the integers $\{1, 2, \dots, N\}$.

The dependence assumed in (2.3) can be given a graph theoretical interpretation. Consider a graph G , with N nodes, labelled as $\{x_1, x_2, \dots, x_N\}$. In this graph, the edge $(x_{m_i}, x_{m_{j(i)}})$, represents the fact the variable x_{m_i} is (statistically) dependent on the variable $x_{m_{j(i)}}$. It is easy to see that G is indeed a tree, with the node x_{m_1} as the root. This tree is completely defined by the permutation $\langle m_1, m_2, \dots, m_N \rangle$ and the function $j(\cdot)$.

It is well known that there are $N^{(N-2)}$ spanning trees on a graph with N nodes. Also, each tree is associated with a unique approximation of the form given in (2.3). The problem of finding the "dependence tree" is one of finding the tree for which the associated approximation is the best.

It is proved in [11] that the closeness measure, $I(P, P_a)$, computed for the product approximation in (2.3) can be expressed as:-

$$I(P, P_a) = - \sum_{i=1}^N I^*(x_{m_i}, x_{m_{j(i)}}) + \sum_{i=1}^N H(x_i) - H(\mathbf{P}) \quad (2.4)$$

where,

$$H(x_i) = - \sum_{x_i} Pr(x_i) \log Pr(x_i)$$

$$H(\mathbf{P}) = - \sum_{\mathbf{X}} P(\mathbf{X}) \log P(\mathbf{X})$$

and

$$I^*(x_i, x_j) = \sum_{x_i, x_j} Pr(x_i, x_j) \log \frac{Pr(x_i, x_j)}{Pr(x_i)Pr(x_j)}$$

The problem of finding the dependence tree, is one of finding the permutation $\langle m_1, m_2, \dots, m_N \rangle$ and the function $j(\cdot)$ (or equivalently an array $(j_1, j_2, \dots, j_N)$) which will minimize the right-hand side of (2.4). Clearly $\sum_{i=1}^N H(x_i)$ and $H(\mathbf{P})$ are independent of the approximation since they only depend on the underlying distribution $P(\mathbf{X})$. Therefore, in order to minimize the RHS of (2.4), we must maximize $\sum_{i=1}^N I^*(x_{m_i}, x_{m_{j(i)}})$.

Although it is not obvious, with a little insight it can be seen that this process is essentially the same as the operation of finding the maximum spanning tree of

the graph G , with N nodes, $x_1, x_2, \dots, x_N$, where the edge between nodes x_i, x_j is assigned the weight $I^*(x_i, x_j)$. In practice, since the probabilities required for computing the edge weights of the graph are not known *a priori*, they must be estimated from the samples. This process is algorithmically described in Program 2.1.

Algorithm DepTree_EMIM_Discrete

Input:

The set of s samples $X^1, X^2, \dots, X^s$.

Output:

The best dependence-tree τ^* as per the EMIM metric.

Method:

- (i) Estimate the first and second-order marginals from the various samples.
- (ii) Compute the edge weights $I^*(x_i, x_j)$ for all pairs of nodes, and create G .
- (iii) Compute the Maximum Spanning Tree of G and submit it as τ^* .

End Algorithm DepTee_EMIM_Discrete

Program 2.1: Chow's Algorithm to Compute Dependence Tree.

It is proved in [11] that Algorithm DepTree_EMIM_Discrete, indeed finds the maximum likelihood estimate of the dependence tree. The significance of this result is tremendous, and although the result was proved in [11], we believe that the power of the result was not adequately stressed. Notice that from the set of all spanning trees of the graph G (totally N^{N-2} of them), the maximum likelihood estimate of the tree is the one which maximizes the likelihood of generating the actual occurrence of samples¹. Thus to obtain such an estimate, the brute force *modus operandus* would suggest that the likelihood function be evaluated for every possible spanning tree, and then the tree which maximized this function would be the reported solution. Considering the large number of spanning trees, this is computationally infeasible except for graphs with a small number of nodes. Algorithm DepTree_EMIM_Discrete

---

¹The fact that the solution space is large does not necessarily imply that the required solution is correspondingly complex.

Figure 2.1: Equivalent procedures for finding the maximum likelihood tree from samples.

is an elegant solution to this problem — it merely involves the straightforward computation of the MST in which the *estimates* of the marginals are used to compute the edge weights. Observe that the operation of finding the Maximum Spanning Tree (MST) has a computational complexity of $O(N^2)$. This result, is stated formally in Theorem 2.1 and is illustrated pictorially in Figure 2.1.

Theorem 2.1 *The dependence-tree produced by the technique Algorithm Dcp-Tree-EMIM-Discrete is the maximum likelihood tree which can be obtained from the samples $X^1, X^2, \dots, X^s$.*

Proof:

A sketch of the proof of this result, was presented in [11]. A complete proof of this result can be found in Appendix A. $\square$

In the light of the above result, it is clear that the metric defined in (1.3) is the best possible metric for capturing dependence information between *pairs* of variables, if the overall approximation is to be ideal in the sense of obtaining the minimum value of $I(P, P_a)$, as defined in (1.2). The only drawback with the I^* measure is

that it is computationally very expensive and time consuming (especially for large values of N), as it involves the evaluation of $O(N^2k^2)$ logarithms where, as before, k is the number of values which each feature of the sample can take. We shall now present a new metric, called the I_χ metric, which essentially captures the χ^2 -statistic of the distributions and which can be used as a measure of dependence between the variables.

2.3 The χ^2 statistic

We define the metric $I_\chi(x_i, x_j)$ to quantify the degree of dependence between two discrete (random) variables as follows:-

$$I_\chi(x_i, x_j) = \sum_{x_i, x_j} \frac{(Pr(x_i, x_j) - Pr(x_i)Pr(x_j))^2}{Pr(x_i)Pr(x_j)} \quad (2.5)$$

From the definition in (2.5), it is clear that I_χ has the following desirable characteristics of a metric capturing dependency information:

1. $I_\chi(x_i, x_j) \geq 0$
2. $I_\chi(x_i, x_j) = 0$ iff $Pr(x_i, x_j) = Pr(x_i)Pr(x_j)$.

Observe that the latter equation represents the case when the variables x_i, x_j are *statistically independent*.

At this juncture, it is worth recapitulating that Section 2.2 concluded with the very important result due to Chow *et. al.* which is that the I^* metric defined in (1.3) naturally appears in the process of selecting the Maximum Likelihood Estimate (MLE) of the dependence tree. This is true, since, by the laws of probability, we are constrained to choose a product form approximation for the joint distribution $P(\mathbf{X})$, and the only metric which minimizes the closeness measure defined in (1.2) is the EMIM given in (1.3).

In this light, it is clear that the metric defined in (2.5), cannot be related to the closeness of approximation defined by (1.2). Furthermore, it does not seem to be

possible to relate this metric to an analogously defined closeness of approximation metric obtained if (2.5) was generalized for the case of distributions as shown below:-

$$I_X(P, P_a) = \sum_{\mathbf{X}} \frac{(P(\mathbf{X}) - P_a(\mathbf{X}))^2}{P_a(\mathbf{X})}$$

Although $I_X(x_i, x_j)$ cannot be related to $I(P, P_a)$ defined in (1.2), it is not entirely futile to pursue it. Observe that for the first part, it is far easier to compute it rather than to compute I^* because as pointed out earlier, the latter involves the evaluation of numerous logarithms. Furthermore, although we cannot prove the optimality of I_X for all distributions, this metric is not entirely void of an underlying theoretical framework. Indeed we can prove the following properties of I_X :

1. In the case when the random variables x_i, x_j are binary valued, the metric $I_X(x_i, x_j)$ defined in (2.5), increases or decreases monotonically with $I^*(x_i, x_j)$.
2. For a restricted class of probability distributions, i.e. for a specific type of dependence between the individual random variables, the quantities $I_X(x_i, x_j)$ and $I^*(x_i, x_j)$ are shown to be equivalent. Thus within this family, the I_X metric indeed yields the MLE of the underlying tree, even though the computation is achieved without computing the matrix $I^*(\cdot, \cdot)$.

Although Property 1 has been proved for the case of binary-valued random variables, we believe that the same result holds even if the restriction of having binary-valued random variables is relaxed. We now prove the above results formally.

Theorem 2.2 *If x_i, x_j are distinct binary-valued random variables, $I_X(x_i, x_j)$ increases(decreases) iff $I^*(x_i, x_j)$ increases (decreases).*

Proof:

We note that both I_X and I^* are functions of the four joint probabilities $Pr(x_i = \alpha, x_j = \beta)$, for $\alpha, \beta = 0, 1$. For the sake of brevity, we denote $p_{\alpha\beta} = Pr(x_i = \alpha, x_j = \beta)$. Since $p_{00} + p_{01} + p_{10} + p_{11} = 1$, all the four $p_{\alpha\beta}$'s cannot be independently specified. Indeed, since any three of them automatically determine the fourth, in the following derivations, we assume p_{00}, p_{01}, p_{10} to be the linearly

independent parameters, and that p_{11} is implicitly specified in terms of their values. We also note that the first order marginals for x_i and x_j can be easily expressed in terms of these second-order marginals. To do this, we introduce the following notation: $a_\alpha = Pr(x_i = \alpha)$ and $b_\alpha = Pr(x_j = \alpha)$. It is easy to see that:

$$\begin{aligned} a_0 &= p_{00} + p_{01} \\ a_1 &= p_{10} + p_{11}, \\ b_0 &= p_{00} + p_{10}, \\ b_1 &= p_{01} + p_{11}. \end{aligned} \quad (2.6)$$

Now, by the definition of I^* in (1.3),

$$\begin{aligned} I^*(x_i, x_j) &= \sum_{i,j} p_{ij} \log \frac{p_{ij}}{a_i b_j} \\ &= p_{00} \log p_{00} - p_{00} \log a_0 b_0 + p_{01} \log p_{01} - p_{01} \log a_0 b_1 \\ &\quad + p_{10} \log p_{10} - p_{10} \log a_1 b_0 + p_{11} \log p_{11} - p_{11} \log a_1 b_1. \end{aligned}$$

Observe that as a consequence of (2.6), $\frac{\partial a_0}{\partial p_{00}} = \frac{\partial b_0}{\partial p_{00}} = 1$ and $\frac{\partial a_1}{\partial p_{00}} = \frac{\partial b_1}{\partial p_{00}} = -1$. Thus, taking partial derivatives with respect to p_{00} , we get,

$$\begin{aligned} \frac{\partial I^*}{\partial p_{00}} &= (\log p_{00} + 1) - (\log(a_0 b_0) + \frac{p_{00}}{a_0 b_0}(a_0 + b_0)) - \frac{p_{01}}{a_0 b_1}(b_1 - a_0) \\ &\quad - \frac{p_{10}}{a_1 b_0}(a_1 - b_0) - (\log p_{11} + 1) - (-\log(a_1 b_1) + \frac{p_{11}}{a_1 b_1}(-a_1 - b_1)). \end{aligned}$$

Simplifying the above equation,

$$\begin{aligned} \frac{\partial I^*}{\partial p_{00}} &= \log \frac{p_{00}/a_0 b_0}{p_{11}/a_1 b_1} + \frac{p_{10} + p_{11}}{a_1} + \frac{p_{01} + p_{11}}{b_1} - \frac{p_{00} + p_{01}}{a_0} - \frac{p_{00} + p_{10}}{b_0} \\ &= \log \frac{p_{00}/a_0 b_0}{p_{11}/a_1 b_1}. \end{aligned} \quad (2.7)$$

We now derive an analogous expression for the derivative for I_χ . Consider,

$$\begin{aligned} I_\chi(x_i, x_j) &= \sum_{i,j} \frac{(p_{ij} - a_i b_j)^2}{a_i b_j} \\ &= \sum_{i,j} \frac{p_{ij}^2}{a_i b_j} - \sum_{i,j} 2p_{ij} + \sum_{i,j} a_i b_j \\ &= \sum_{i,j} \frac{p_{ij}^2}{a_i b_j} - 1. \end{aligned} \quad (2.8)$$

Using the simplified form of I_x in (2.8), and taking the partial derivative w.r.t. p_{00} , we get,

$$\begin{aligned}\frac{\partial I_x}{\partial p_{00}} &= 2 \frac{p_{00}}{a_0 b_0} - \left(\frac{p_{00}}{a_0 b_0} \right)^2 (a_0 + b_0) \\ &\quad - \left(\frac{p_{01}}{a_0 b_1} \right)^2 (b_1 - a_0) - \left(\frac{p_{10}}{a_1 b_0} \right)^2 (a_1 - b_0) - 2 \frac{p_{11}}{a_1 b_1} \\ &\quad - \left(\frac{p_{11}}{a_1 b_1} \right)^2 (-a_1 - b_1).\end{aligned}$$

Grouping terms and performing some elementary algebraic simplifications (e.g. $a_1 - b_0 = b_1 - a_0$, $a_1 + b_1 = 2 - (a_0 + b_0)$), we obtain,

$$\begin{aligned}\frac{\partial I_x}{\partial p_{00}} &= 2 \left(\frac{p_{00}}{a_0 b_0} - \frac{p_{11}}{a_1 b_1} \right) - (a_1 - b_0) \left\{ \left(\frac{p_{01}}{a_0 b_1} \right)^2 + \left(\frac{p_{10}}{a_1 b_0} \right)^2 \right\} \\ &\quad + 2 \left(\frac{p_{11}}{a_1 b_1} \right)^2 - (a_0 + b_0) \left\{ \left(\frac{p_{00}}{a_0 b_0} \right)^2 + \left(\frac{p_{11}}{a_1 b_1} \right)^2 \right\}.\end{aligned}$$

To simplify the above expression, we define r_{ij} as:

$$r_{ij} = \frac{p_{ij}}{a_i b_j}.$$

Thus the above equation (for the derivative) can be conveniently rewritten as:

$$\frac{\partial I_x}{\partial p_{00}} = 2(r_{00} - r_{11}) - (a_1 - b_0)(r_{01}^2 + r_{10}^2) - (a_0 + b_0)(r_{00}^2 + r_{11}^2) + 2r_{11}^2. \quad (2.9)$$

We shall now derive certain auxiliary equations, relating the $\{r_{ij}\}$'s. Consider the difference $r_{00} - r_{11}$. Indeed, this has the value,

$$\begin{aligned}r_{00} - r_{11} &= \frac{p_{00}}{a_0 b_0} - \frac{p_{11}}{a_1 b_1} = \frac{p_{00} a_1 b_1 - p_{11} a_0 b_0}{a_0 a_1 b_0 b_1} \\ &= \frac{p_{00} \{p_{11}^2 + (p_{01} + p_{10})p_{11} + p_{01}p_{10}\} - p_{11} \{p_{00}^2 + (p_{01} + p_{10})p_{00} + p_{01}p_{10}\}}{a_0 a_1 b_0 b_1} \\ &= \frac{(p_{11} - p_{00})D}{a_0 a_1 b_0 b_1} \quad \text{where } D = p_{00}p_{11} - p_{01}p_{10}.\end{aligned} \quad (2.10)$$

In a similar manner, we can obtain the following equations for $r_{00} - r_{01}$ as:

$$\begin{aligned} r_{00} - r_{01} &= \frac{p_{00}}{a_0 b_0} - \frac{p_{01}}{a_0 b_1} = \frac{p_{00} b_1 - p_{01} b_0}{a_0 b_0 b_1} \\ &= \frac{p_{00}(p_{11} + p_{01}) - p_{01}(p_{10} + p_{00})}{a_0 b_0 b_1} \\ &= D/a_0 b_0 b_1. \end{aligned}$$

Using the symmetry of the situation, the rest of the auxiliary relations between $\{r_{ij}\}$'s can be written down as:

$$\begin{aligned} r_{00} - r_{10} &= D/(a_0 a_1 b_0) \\ r_{11} - r_{01} &= D/(a_0 a_1 b_1) \\ r_{11} - r_{10} &= D/(a_1 b_0 b_1) \end{aligned} \quad (2.11)$$

We note that $a_0 + b_0 = 1 - (a_1 - b_0)$, and hence we can rewrite the expression in (2.9) for $\frac{\partial I_x}{\partial p_{00}}$ as:

$$\begin{aligned} \frac{\partial I_x}{\partial p_{00}} &= 2(r_{00} - r_{11}) - (a_1 - b_0)(r_{01}^2 + r_{10}^2) + (a_1 - b_0)(r_{00}^2 + r_{11}^2) \\ &\quad - (r_{00}^2 + r_{11}^2) + 2r_{11}^2 \\ &= (r_{00} - r_{11})(2 - r_{00} - r_{11}) + (a_1 - b_0)(r_{00}^2 + r_{11}^2 - r_{01}^2 - r_{10}^2) \\ &= (r_{00} - r_{11})(2 - r_{00} - r_{11}) + (p_{11} - p_{00}) \left\{ \frac{D}{a_0 b_0 b_1} (r_{00} + r_{01}) \right. \\ &\quad \left. + \frac{D}{a_1 b_0 b_1} (r_{11} + r_{10}) \right\} \\ &= (r_{00} - r_{11})(2 - r_{00} - r_{11}) + \frac{D(p_{11} - p_{00})}{a_0 a_1 b_0 b_1} \{a_1(r_{00} + r_{01}) + a_0(r_{11} + r_{10})\} \\ &= (r_{00} - r_{11}) \{2 - r_{00} - r_{11} + a_1(r_{00} + r_{01}) + a_0(r_{11} + r_{10})\} \\ &= (r_{00} - r_{11}) \{2 + a_1(r_{01} - r_{11}) + a_0(r_{10} - r_{00})\}. \end{aligned} \quad (2.12)$$

Consider the second term of the RHS of (2.12). We can simplify this using the values of $\{r_{ij}\}$ and the relation between the $\{r_{ij}\}$ (2.11) to yield:

$$\begin{aligned} 2 + a_1(r_{01} - r_{11}) + a_0(r_{10} - r_{00}) &= 2 - \frac{a_1 D}{a_0 a_1 b_1} - \frac{a_0 D}{a_0 a_1 b_0} \\ &= 2 - \frac{D}{a_0 b_1} - \frac{D}{a_1 b_0} \end{aligned}$$

$$= \left(1 - \frac{D}{a_0 b_1}\right) + \left(1 - \frac{D}{a_1 b_0}\right). \quad (2.13)$$

Also since,

$$\begin{aligned} a_0 b_1 &= (p_{00} + p_{01})(p_{01} + p_{11}) \\ &= p_{01}^2 + (p_{00} + p_{11})p_{01} + p_{00}p_{11} \end{aligned}$$

$$\text{it is clear that } a_0 b_1 - D = p_{01}^2 + (p_{00} + p_{11})p_{01} + p_{01}p_{10} = p_{01}$$

$$\text{similarly } a_1 b_0 - D = p_{10}.$$

Hence, combining (2.12) and (2.13), we get:

$$\begin{aligned} \frac{\partial I_x}{\partial p_{00}} &= (r_{00} - r_{11}) \left\{ \left(1 - \frac{D}{a_0 b_1}\right) + \left(1 - \frac{D}{a_1 b_0}\right) \right\} \\ &= (r_{00} - r_{11}) \left\{ \frac{p_{01}}{a_0 b_1} + \frac{p_{10}}{a_1 b_0} \right\} \\ &= (r_{00} - r_{11})(r_{01} + r_{10}). \end{aligned} \quad (2.14)$$

Comparing equations (2.7) and (2.14), we see that for all distinct random variables, x_i, x_j , $\frac{\partial I_x}{\partial p_{00}}$ always follows $\frac{\partial I^*}{\partial p_{00}}$ in sign. Thus the theorem is proved with regard to the variation of the quantities w.r.t. p_{00} .

We now derive an expression for the derivatives of I^* and I_x w.r.t. the variable p_{01} . By the definition of I^* in (1.3), we can write:

$$I^*(x_i, x_j) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{a_i b_j}.$$

Because of (2.6), we immediately deduce that $\frac{\partial a_0}{\partial p_{01}} = 1$ and $\frac{\partial b_0}{\partial p_{01}} = 0$ and therefore, differentiating I^* w.r.t. p_{01} , we obtain:

$$\begin{aligned} \frac{\partial I^*}{\partial p_{01}} &= -\frac{p_{00}}{a_0 b_0} (b_0 \cdot 1) + (1 + \log p_{01}) - \left\{ \frac{p_{01}}{a_0 b_1} (b_1 \cdot 1) + \log(a_0 b_1) \right\} \\ &\quad - \frac{p_{10}}{a_1 b_0} (b_0)(-1) \\ &\quad + (1 + \log p_{11})(-1) - \left\{ (-1) \log(a_0 b_1) + \frac{p_{11}}{a_1 b_1} (-b_1) \right\} \\ &= -\frac{p_{00}}{a_0} + 1 + \log p_{01} - \frac{p_{01}}{a_0} - \log(a_0 b_1) \\ &\quad + \frac{p_{10}}{a_1} - 1 - \log p_{11} + \log(a_1 b_1) + \frac{p_{11}}{a_1}. \end{aligned}$$

$$\begin{aligned}
&= \log \frac{p_{01}/a_0 b_1}{p_{11}/a_1 b_1} + \frac{p_{10} + p_{11}}{a_1} - \frac{p_{00} + p_{01}}{a_0} \\
&= \log \frac{r_{01}}{r_{11}}.
\end{aligned}$$

We now proceed to differentiate I_x w.r.t. p_{01} . Using the simplified form of I_x given in (2.8), we obtain:

$$\begin{aligned}
\frac{\partial I_x}{\partial p_{01}} &= -\left(\frac{p_{00}}{a_0 b_0}\right)^2 (b_0) + 2\frac{p_{01}}{a_0 b_1} - \left(\frac{p_{01}}{a_0 b_1}\right)^2 (b_1) \\
&\quad - \left(\frac{p_{10}}{a_1 b_0}\right)^2 (-b_0) - 2\frac{p_{11}}{a_1 b_1} - \left(\frac{p_{11}}{a_1 b_1}\right)^2 (-b_1) \\
&= 2(r_{01} - r_{11}) - b_0 r_{00}^2 - b_1 r_{01}^2 + b_0 r_{10}^2 + b_1 r_{11}^2 \\
&= 2(r_{01} - r_{11}) - b_0(r_{00}^2 - r_{10}^2) + b_1(r_{11}^2 - r_{01}^2) \\
&= 2(r_{01} - r_{11}) - \frac{b_0(r_{00} + r_{10})D}{a_0 a_1 b_0} + \frac{b_1(r_{11} + r_{01})D}{a_1 a_0 b_1} \\
&= 2(r_{01} - r_{11}) + \frac{D}{a_0 a_1} [r_{11} + r_{01} - r_{00} - r_{10}]
\end{aligned}$$

Now, from (2.11), we can write:

$$\frac{D}{a_0 a_1} = -b_1(r_{01} - r_{11}).$$

and hence, $\frac{\partial I_x}{\partial p_{01}}$ can be further simplified to be:

$$\begin{aligned}
\frac{\partial I_x}{\partial p_{01}} &= (r_{01} - r_{11}) [2 - b_1(r_{11} + r_{01} - r_{00} - r_{10})] \\
&= (r_{01} - r_{11}) [2 - b_1 \{(r_{11} - r_{10}) + (r_{01} - r_{00})\}] \\
&= (r_{01} - r_{11}) \left[2 - b_1 \left\{ \frac{D}{a_1 b_0 b_1} - \frac{D}{a_0 b_0 b_1} \right\} \right] \\
&= (r_{01} - r_{11}) \left[2 - \frac{D}{a_1 b_0} + \frac{D}{a_0 b_0} \right] \\
&= (r_{01} - r_{11}) \left[\left(1 - \frac{D}{a_1 b_0} \right) + \left(1 + \frac{D}{a_0 b_0} \right) \right]
\end{aligned}$$

Observe that we had derived that $a_1 b_0 - D = p_{10}$. This relation immediately yields that the first quantity in the square brackets is simply $p_{10}/(a_1 b_0)$ or r_{10} .

To evaluate the second quantity within the square brackets, we consider the term $a_0 b_0 + D$. Using (2.6), this quantity can be written as:

$$\begin{aligned}
 a_0 b_0 + D &= (p_{00} + p_{01})(p_{00} + p_{10}) + D \\
 &= p_{00}(p_{00} + p_{01} + p_{10}) + p_{01}p_{10} + p_{00}p_{11} - p_{01}p_{10} \\
 &= p_{00}(p_{00} + p_{01} + p_{10}) + p_{00}p_{11} \\
 &= p_{00}(p_{00} + p_{01} + p_{10} + p_{11}) \\
 &= p_{00}.
 \end{aligned}$$

Hence, the quantity $1 + \frac{D}{a_0 b_0}$ can be easily seen to be $p_{00}/(a_0 b_0)$ or r_{00} . Therefore, $\frac{\partial I_x}{\partial p_{01}}$ can be written as follows:

$$\frac{\partial I_x}{\partial p_{01}} = (r_{01} - r_{11})(r_{10} + r_{00}).$$

Similar expressions can be derived for the partial derivatives of I_x and I^* , w.r.t. the variable p_{10} . In the interest of brevity, we do not derive these expressions, but merely but summarize the results for the derivatives w.r.t. p_{01} and p_{10} below:-

$$\begin{aligned}
 \frac{\partial I_x}{\partial p_{01}} &= (r_{01} - r_{11})(r_{10} + r_{00}) \\
 \frac{\partial I^*}{\partial p_{01}} &= \log \frac{r_{01}}{r_{11}} \\
 \frac{\partial I_x}{\partial p_{10}} &= (r_{10} - r_{11})(r_{01} + r_{00}) \\
 \frac{\partial I^*}{\partial p_{10}} &= \log \frac{r_{10}}{r_{11}}.
 \end{aligned} \tag{2.15}$$

An inspection of the equations in (2.15) reveals that the partial derivatives of I_x and I^* w.r.t. p_{01} and p_{10} have the same sign. Hence the Theorem. $\square$

Remark:

A comparison of Equations (2.7) and (2.14) indicates the equality of sign only if r_{01} and r_{10} are *both* non-zero. It is interesting to note the implication of this condition. Observe that if r_{01} and r_{10} are both zero, the definition of r_{ij} implies that $p_{01} = p_{10} = 0$, and that the values of x_i, x_j are closely related (indeed, they

are *never* different). In this case the condition $r_{01} = r_{10} = 0$, is equivalent to the condition that x_i is *identically equal* to x_j and thus x_i and x_j are non-distinct.

Furthermore, an observation of (2.15) indicates that the derivatives of I_x and I^* w.r.t. p_{01} agree in sign everywhere, except when $r_{10} = r_{00} = 0$. This condition implies that $p_{10} = p_{00} = 0$, or that $b_0 = 0$. The condition $b_0 = 0$ implies that the random variable x_j is always 1, and hence is not informative. Such random variables can therefore be assumed to be absent. Similar arguments can be made with regard to the partial derivative w.r.t. p_{10} . In summary, except in trivial cases, the partial derivatives of I_x and I^* always agree in sign.

The result established in Theorem 2.2 implies that if, as a result of slightly perturbing the second-order marginals of the variables x_i and x_j , the measure $I^*(x_i, x_j)$ increases or decreases, a similar effect will be observed in the measure $I_x(x_i, x_j)$. Though the result is all-encompassing and powerful, it is unfortunately quite a local result. Thus, although the theorem guarantees how the weight assigned to an edge will vary depending on the metric used, it does not guarantee how the weight of the *MST* will vary as a function of the metric used. In other words, this property does not ensure that these two metrics are “globally” equivalent when it concerns computing the *MST*. In order to guarantee that both measures of dependencies find the same dependency tree, the relative ordering of the weights for **all** the edges meeting at a particular node must be preserved, irrespective of the metric used (i.e. I_x or I^*).

Although I^* and I_x only locally follow one another for all distributions, there is a subset of distributions for which there is a global relationship between the two. Indeed, if the second-order marginals of the distributions satisfy certain elementary constraints, we can show that I^* and I_x globally follow each other. Thus working with a restricted set of distributions, it becomes evident that the *MST* chosen using the I_x metric is **exactly** the *MLE* for the underlying tree. The following sequence of Lemmas prove the result.

Lemma 2.1 *Consider the pair of variables x_i, x_j and the joint probabilities defined as follows:*

$$\begin{aligned} p_{00} &= a - \lambda \\ p_{01} &= b + \lambda \\ p_{10} &= c + \lambda \\ p_{11} &= d - \lambda \end{aligned} \tag{2.16}$$

where a, b, c, d and λ satisfy:

$$\begin{aligned} a + b + c + d &= 1 \\ ad &= bc \\ 0 &\leq \lambda \leq \min(a, d) \end{aligned} \tag{2.17}$$

Then $dI^*/d\lambda$ follows $dI_x/d\lambda$ in sign.

Proof:

We note that from (2.8):

$$I_x(x_i, x_j) = \sum_{i,j} \frac{p_{ij}^2}{a_i b_j} - 1.$$

Now, the total derivative of I_x can be written as:

$$\begin{aligned} \frac{dI_x}{d\lambda} &= \frac{2p_{00}(-1)}{a_0 b_0} + \frac{2p_{01}}{a_0 b_1} + \frac{2p_{10}}{a_1 b_0} + \frac{2p_{11}(-1)}{a_1 b_1} \\ &= 2 \frac{(\lambda - a)b_1 + (b + \lambda)b_0}{a_0 b_0 b_1} + 2 \frac{(c + \lambda)b_1 + (\lambda - d)b_0}{a_1 b_0 b_1} \end{aligned} \tag{2.18}$$

The numerator of the first term of the RHS of (2.18) is:

$$\begin{aligned} (\lambda - a)b_1 + (b + \lambda)b_0 &= \lambda(b_0 + b_1) - a(b + d) + b(c + a) \\ &= \lambda - (ad - bc) = \lambda, \end{aligned}$$

the last equality being a consequence of (2.17).

Similarly, the numerator of the second term of the RHS of (2.18) is:

$$(\lambda - d)b_0 + (c + \lambda)b_1 = \lambda - d(c + a) + c(b + d) = \lambda$$

Thus,

$$\begin{aligned}\frac{dI_x}{d\lambda} &= \frac{2\lambda}{a_0 b_0 b_1} + \frac{2\lambda}{a_1 b_0 b_1} = \frac{2\lambda(a_0 + a_1)}{a_0 a_1 b_0 b_1} \\ &= \frac{2\lambda}{a_0 a_1 b_0 b_1},\end{aligned}\tag{2.19}$$

where again the last equality follows since $a_0 + a_1 = 1$.

We now proceed to evaluate the derivative of I^* w.r.t. λ . Since,

$$\begin{aligned}I^*(x_i, x_j) &= \sum_{i,j} p_{ij} \log \frac{p_{ij}}{a_i b_j} \\ &= \sum p_{ij} \log p_{ij} - \sum p_{ij} \log K_{ij}, \quad \text{where } K_{ij} = a_i b_j,\end{aligned}$$

we have

$$\begin{aligned}\frac{dI^*}{d\lambda} &= (1 + \log p_{00})(-1) + (1 + \log p_{01}) + (1 + \log p_{10}) + (1 + \log p_{11})(-1) \\ &\quad + \log K_{00} + \log K_{11} - \log K_{01} - \log K_{10} \\ &= \log \frac{p_{01} p_{10}}{p_{00} p_{11}} + \log \frac{K_{00} K_{11}}{K_{01} K_{10}}.\end{aligned}$$

Now it is easy to verify that $K_{00} K_{11} / K_{01} K_{10} = 1$. Hence the second term of the equation for $dI^*/d\lambda$ vanishes. Thus,

$$\begin{aligned}\frac{dI^*}{d\lambda} &= \log \frac{p_{01} p_{10}}{p_{00} p_{11}} \\ &= \log \frac{(b + \lambda)(c + \lambda)}{(a - \lambda)(d - \lambda)} \\ &= \log \frac{\lambda^2 + (b + c)\lambda + bc}{\lambda^2 - (a + d)\lambda + ad} \\ &= \log \frac{\lambda^2 + (1 - \alpha)\lambda + \beta}{\lambda^2 - \alpha\lambda + \beta} \quad \text{where } \alpha = a + d, \text{ and } \beta = ad\end{aligned}\tag{2.20}$$

Comparing (2.19) and (2.20), we see that both $dI_x/d\lambda$ and $dI^*/d\lambda$ are positive, negative or zero, according as λ is positive, negative, or zero respectively. Hence we have proved Lemma 1. $\square$

Lemma 2.2 *Let x_i be any random variable. Then for all $j \neq i$, let the joint distribution of the pair of random variables x_i, x_j satisfy the conditions (2.16) and (2.17) of Lemma 1 with the additional constraint that all distributions are based on common values of the parameters a, b, c, d but differ only in the value of the parameter λ . Then the index k which maximizes $I^*(x_i, x_j)$ also maximizes $I_x(x_i, x_j)$.*

Proof:

Since all the edges have common values of the parameters a, b, c, d , but differ only in the value of λ , the edge weights (under I^* and I_λ metrics) are functions of the same single parameter λ . The stated result simply follows from Lemma 1 as per the observation that both I^* and I_λ are both increasing functions of λ in the interval that λ is constrained to belong to. This implies that the index of the edge with maximum weight remains unchanged under both metrics. $\square$

Theorem 2.3 *If the joint distributions for every pair of variables x_i, x_j have the same parametric form defined by (2.16) and (2.17) with identical values of a, b, c, d , but the distributions differ only in the parameter λ , then the MST chosen by I_λ will be exactly identical to that chosen by I^* or is just as good an approximation as the latter.*

Proof:

Lemma 2.2 has established the result that of the edges meeting at a particular node, the index of edge with maximum weight is invariant under both the metrics I^* and I_λ . Given the additional information that **all edges** are specified using the same parametric constants (i.e. a, b, c, d), a straightforward extension of Lemma 2.2 yields the result that the **relative ordering** among the $\binom{N}{2}$ $I^*(\cdot, \cdot)$ weights and $I_\lambda(\cdot, \cdot)$ weights are identical.

Several algorithms to solve the MST problem are known. We shall show the equivalence of the trees produced by I^* and I_λ metrics, by simulating Kruskal's algorithm [2]. This algorithm essentially sorts the edges in the descending order of weights and selects the $(N - 1)$ edges that do not form cycles. It is apparent that since the relative ordering of the edge weights under the two metrics are identical, the MST computed will be identical.

It must be noted however that the MST is unique only if all the edge weights are distinct. If two edges have identical weights under I^* metric, they can be easily shown to have identical weights under the I_λ metric. Thus in these cases, the MST's produced by the two metrics can only differ in the selection of an edge from

the set of edges with equal weight. The sum of the weights of the edges included in the MST will indeed be identical and hence both the MSTs will be equally good approximations (see (2.4)). $\square$

It is interesting to note that the decision to specify all the $\binom{N}{2}$ distributions identically, has other implications. These conditions are indicated below:

Lemma 2.3 *If all the $\binom{N}{2}$ pairs of variables in the graph G use the same parameters a, b, c, d , then $b = c$ and the values of a, d are given by: $\{(1 - 2b) \pm \sqrt{1 - 4b}\}/2$.*

Proof:

Let us consider a node i and its neighbor j . Then $Pr(x_i) = 0$ and $Pr(x_j) = 0$ are given by $a + b$ and $a + c$ respectively. Because of the symmetry, we must have $a + b = a + c$, and hence $b = c$. Given the additional conditions $a + b + c + d = 1$ and $ad = bc$, we can solve for the two unknowns (i.e. a and d) from these two equations. The above equation indicates that $b \leq 0.25$. When $b = 0.25$, all the constants a, b, c, d are equal (to 0.25). $\square$

2.4 Experimental Results

In order to test the validity of the theoretical results presented in this chapter, numerous simulations were carried out. The experiments conducted were of two categories. In the first set of experiments the intention was primarily to test the accuracy of the new metric I_x . That is, in this case the aim was to determine the number of times the MST τ_x (obtained by using I_x) was identical to the corresponding tree τ^* (derived by I^*). In the second set of experiments, the aim was to observe how quickly the τ_x and τ^* converged to the "real" underlying tree.

As a consequence of Theorems 2.2 and 2.3, it is easy to see that whenever the conditions of (2.16) and (2.17) are satisfied and the random variable distributions have identical parameters, both the I_x and the I^* metric will be equally accurate. Of course this has been experimentally verified and in every single case, when the conditions were satisfied, τ_x and τ^* were either identical or equally efficient w.r.t.

the EMIM metric I^* , where by “equally efficient” we mean that the sum of the I^* weights for all the edges in τ_χ and τ^* are identical. Thus any further simulations done for this scenario can only further justify the assertion.

The quality of the estimate τ_χ can be evaluated for the case when the conditions of Theorem 2.3 are not satisfied. Let us suppose that some (or all) pairs of random variables x_i, x_j have a second-order marginal distribution $Pr(x_i, x_j)$, which does not satisfy these conditions. Then the estimate τ_χ need not be identical to (or equally efficient as) τ^* . However we have observed experimentally that for a good proportion of time, τ_χ is identical to τ^* and in the cases when they are not identical, the relative difference between the weights of τ_χ and τ^* (i.e. sum of the weights of all the edges) as per the EMIM metric I^* is extremely small — being of the order of 0.5%.

These experiments were conducted for various dimensions of the random vector, as follows. First the number of components was selected and each second-order marginal distribution was randomly generated. The procedure by which these are generated is easy to follow, if we consider the following sub-problem. Let us limit ourselves to dealing with two events A and B , which occur with probabilities $Pr(A)$ and $Pr(B)$ respectively. The problem which we wish to address is one of randomly assigning a probability to the joint event $A \cap B$.

In order to assign probabilities to the event $A \cap B$, we have to first establish some bounds for this quantity. Clearly $0 \leq Pr(A \cap B) \leq \min(Pr(A), Pr(B))$. Furthermore, since,

$$Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$$

and $Pr(A \cup B) \leq 1$,

we easily get the following inequalities:

$$\begin{aligned} Pr(A \cap B) &= Pr(A) + Pr(B) - Pr(A \cup B) \\ &\geq Pr(A) + Pr(B) - 1. \end{aligned}$$

Combining the above results, we arrive at the following relation:

$$\max(0, Pr(A) + Pr(B) - 1) \leq Pr(A \cap B) \leq \min(Pr(A), Pr(B)). \quad (2.21)$$

Given the lower and upper bounds established by this equation, it is easy to randomly assign a value to $Pr(A \cap B)$.

We now illustrate the use of the above result, in assigning the second-order marginal distributions for the pair of random variables x_i and x_j . For simplicity, we assume that these variables are binary valued. The first step in this process is to assign the first-order marginals to the random variables x_i and x_j . In other words we first uniformly assign values to $Pr(x_i = 0)$ and $Pr(x_j = 0)$, and thus implicitly assign values to $Pr(x_i = 1)$ and $Pr(x_j = 1)$. Using the bounds specified by (2.21), we can now randomly assign a value to the joint probability $Pr(x_i = 0, x_j = 0)$. Once this choice has been made, the remaining second-order marginals can be written down as follows:

$$Pr(x_i = 0, x_j = 1) = Pr(x_i = 0) - Pr(x_i = 0, x_j = 0),$$

$$Pr(x_i = 1, x_j = 0) = Pr(x_j = 0) - Pr(x_i = 0, x_j = 0),$$

$$Pr(x_i = 1, x_j = 1) = Pr(x_i = 1) - Pr(x_i = 0, x_j = 1).$$

To get the joint distribution for all the random variables, the above procedure was repeated for all pairwise combinations of variables.

We are now ready to present the results obtained for the above mentioned case. The results represent the ensemble average, computed over 5000 random distributions, for each value of the dimension D of the random vector. In the following discussion $EMIM(\tau)$ stands for $\sum_{e \in \tau} I^*(e)$. The two parameters we report are (i) the number of times τ^* and τ_x are different, and (ii) the Relative Error between $EMIM(\tau^*)$ and $EMIM(\tau_x)$. This quantity is computed as the percentage average of $(EMIM(\tau^*) - EMIM(\tau_x))/EMIM(\tau^*)$. Table 2.1 presents our results. The results are significant. For example, when N the number of dimensions is 6, the percentage error between the EMIM values of τ^* and τ_x is as low as 0.277%. This value increases to 0.49 %, when the dimension N increases to 12.

Similar experiments were conducted in the case when the random variables were not binary valued, but were ternary valued (3-valued). Table 2.2 presents the results

Dimension of Feature Vector (N)	Proportion of Mismatches (in %)	Average Relative Error (in %)
6	22.12	0.2771
8	39.02	0.3799
10	52.28	0.4470
12	62.58	0.4912

Table 2.1: This table summarizes the results of comparing the trees produced by the I_x and I^* metrics, when random distributions were used. The reported results are the number of mismatches and the accuracy of the tree weights. In each case, the random variables are binary valued.

Dimension of Feature Vector (N)	Proportion of Mismatches (in %)	Average Relative Error (in %)
6	46.34	1.2941
8	65.62	1.6487
10	79.54	1.8448

Table 2.2: This table summarizes the results of comparing the trees produced by the I_x and I^* metrics, when random distributions were used. The reported results are the number of mismatches and the accuracy of the tree weights. In each case (i.e. for each value of N), each of the random variables take one of 3 values.

for this case. Notice that although the number of mismatches is relatively large, the percentage error between the weights of two trees τ^* and τ_x is small.

In the second set of experiments, instead of generating the probability distributions directly, we used an underlying tree to generate the samples. Given the dependence tree, the program must generate the various components of the vector in such a way that the "parent" variable is assigned a value before the "child" (or dependent) variable. It is easy to see that this condition is satisfied if the random variables are assigned values in the order that a breadth-first traversal of the dependence tree would visit them. The estimates of the second-order (and the first-order)

marginals were updated with every sample. Subsequent to an initial phase, the trees τ_x and τ^* were computed as the samples arrived. As before, we maintain the number of times the trees τ^* and τ_x were different. Since the dependence in this case comes from an **unknown underlying tree** τ_r , it is useful to compare the convergence of τ^* and τ_x to this tree τ_r . One measure of the closeness of these trees is the closeness of the distributions generated by them and this is precisely what we chose to monitor. If P_r , P^* and P_x are the distributions derived from the “real” underlying tree τ_r , τ^* and τ_x respectively, the quantities which we measured are the information theoretic closeness measures $I(P_r, P_x)$ and $I(P_r, P^*)$. Note that these quantities can be easily computed from the definition given in (1.2).

We now present the results for the case when N , the number of dimensions, is 6. The underlying dependence tree used is shown in Figure 2.2a. In our experiments, the agreement between τ^* and τ_x was so close that the plots of $I(P_r, P_x)$ and $I(P_r, P^*)$ vs. n , the number of samples, can hardly be differentiated visually. For this reason, in Figure 2.2b we have shown $I(P_r, P_x)$ as a function of n . The closeness of approximation in Figure 2.2b, was obtained as the ensemble average over 50 experiments, each consisting of 5000 samples. Besides the fact that $I(P_r, P_x)$ and $I(P_r, P^*)$ are extremely close, two other observations are also worth mentioning. When the underlying dependence is one of tree-type dependence, the agreement between τ^* and τ_x is indeed very close. This is evidenced by a very small number of mismatches when compared to the case of random distributions. The other observation is that in terms of the number of samples examined to converge to the best tree, both metrics seem to be comparable. For the case illustrated, the number of samples required to reduce the “initial distance” between the real tree and the estimate to 10% of its value, is 1300. It must however be kept in mind that even though the “real dependency” tree may have been found earlier, the closeness measure between these distributions is not exactly zero, due to **inaccurate estimates** of the conditional probabilities. Figure 2.3 reports similar results for the case when N , the dimensionality of the random vector, is 10.

In order to quantify the ease of computing τ_x over that of τ^* , we have performed

(a) Underlying dependence tree used for generating the samples to measure the rate of convergence of τ_x . In this case, N , the dimensionality of the random vector is 6, which is equal to the number of nodes in this tree.

(b) Variation of the ensemble average of the closeness of approximation $I(P, P_a)$ between the actual distribution P and the estimated distribution P_a derived from the estimated dependence tree. The metric used to derive this tree is the I_x metric. The underlying dependence tree describing P is given in Figure 2.2a.

Figure 2.2: Rate of Convergence. Discrete Vector case ($N=6$).

(a) Underlying dependence tree used for generating the samples to measure the rate of convergence of τ_x . In this case, N , the dimensionality of the random vector is 10, which is equal to the number of nodes in this tree.

(b) Variation of the ensemble average of the closeness of approximation $I(P, P_a)$ between the actual distribution P and the estimated distribution P_a derived from the estimated dependence tree. The metric used to derive this tree is the I_x metric. The underlying dependence tree describing P is given in Figure 3a.

Figure 2.3: Rate of Convergence. Discrete Vector case ($N=10$).

some measurements on the execution times ². The experiment consisted of generating a random probability distribution for the case of binary valued random vectors and then computing each of τ^* and τ_x 5000 times each. The figures on CPU time usage suggest that the computation of τ_x takes approximately 22% of the time required to compute τ^* . This reduction in computation time is a natural consequence of not requiring the evaluation of a large number of logarithmic terms in the computation of the edge weight matrix. Indeed, in terms of real computations, the effort required to compute I_x is almost an order of magnitude less than the corresponding effort involved in computing τ^* . It can be seen that the savings in terms of CPU time, will be even more significant if the number of values that each feature can assume, increases.

---

²The reader should note that both Chow's and our algorithm require computations of $O(N^2)$ weights and the subsequent computation of the MST. The computational gain we obtain is primarily because whereas our weights involve quadratic terms, Chow's involve $O(N^2k^2)$ logarithmic terms.

2.5 Summary

In this chapter, we have considered the problem of approximating an unknown underlying discrete probability distribution, by one derived from a dependence tree. The best dependence tree τ^* is known to be the MST of a complete graph, with $I^*(x_i, x_j)$, the information theoretic metric, as the edge weight between the pair of nodes x_i and x_j . This chapter proposes a chi-squared based metric, I_χ to capture the dependence information between pairs of random variables, and the tree generated using I_χ is referred to as τ_χ .

If we restrict ourselves to binary random variables only, the quantities I^* and I_χ are shown to follow one another locally, i.e. they increase or decrease together. We conjecture that this result can be generalized for non-binary random vectors. By suitably restricting the domain from which the joint probabilities can be assigned to pairs of random variables, we have shown that both these metrics are equivalent (and hence optimal) in this restricted world.

Experimental results clearly demonstrate that when the underlying unknown distribution is derived from a dependence tree, both metrics I^* and I_χ succeed in finding it. When the underlying dependence is not actually based on a tree, both τ^* and τ_χ are estimates for the best dependence tree. Even in this case, whenever the two estimates of the best dependence tree do not exactly match, their total weights are almost indistinguishably close.

The main advantage with our metric I_χ is that it is computationally far superior to the metric I^* ; this is a direct consequence of not requiring the evaluation of numerous logarithms. The experimental results suggest that τ_χ can be computed in a small fraction of the time required to compute τ^* . The savings in terms of CPU time, will be even more significant, if the number of values that each random variable can assume is increased.

A summary of these results have been published [91].

Chapter 3

Determining Dependence Trees for Continuous Random Vectors

Having described the use of the χ^2 metric for the discrete random vector case, we shall now proceed to consider the case when the vectors are continuous. Let $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$ be an N -dimensional continuous random vector. As before, the basic problem that we wish to address is one of estimating the joint probability density $P(\mathbf{X})$, from measurements on the vector itself. Due to the fact that the number of samples may be small, and that real machines often impose limitations on available storage, it is often necessary to restrict the amount of information that can be collected from the samples. Keeping this constraint in mind, as mentioned earlier, it is common to make various simplifying assumptions about the distribution $P(\mathbf{X})$, and the simplest assumption made about this distribution is that the random variables or features are statistically independent. This assumption, namely that regarding stochastic independence, typically simplifies the solution that is obtained. Clearly whenever this assumption does not model the underlying real life situation adequately, the dependence information has to be incorporated in the actual model. This aspect was explained for the discrete variable case but since the scenario changes considerably for the continuous case we shall clarify it when the random vector is jointly normal.

Let us consider a pattern recognition scheme, and focus on the aspect of analyzing the features of only one class. Since the features may be completely dependent,

in general, the covariance matrix Σ may be full. However, by redefining the features so that the feature vector is a linear combination of the original vector, it can be shown that the transformed set of features can be made statistically independent [22, pp.31-35]. Indeed, the linear transformation essentially involves pre-multiplying the original feature vector by the matrix of eigenvectors of Σ , and since Σ is symmetric and positive definite, these eigenvectors will be orthonormal and the resulting covariance matrix of the transformed feature vector will be diagonal.

The question is a little more complicated when we are studying the features of two classes, say ω_1 and ω_2 . Let us suppose that the class-conditional densities of these classes have covariance matrices Σ_1 and Σ_2 . Using a transformation analogous to the one described above, it is easy to diagonalize Σ_1 (i.e., make the features of class ω_1 independent) while the features of the class ω_2 are still dependent. Let us denote the transformed covariance matrices as Σ'_1 and Σ'_2 respectively. By appropriately scaling the transformed features of the class ω_1 , it is possible to render it to have the identity matrix as its covariance matrix. If the new covariance matrices are Σ''_1 and Σ''_2 , then, this operation renders $\Sigma''_1 = I$. Subsequently the axes of the features can be "rotated" so as to render them parallel to the eigenvectors of the covariance matrix Σ''_2 . Thus even in the two class problem, after the appropriate "rotation" and scaling operations, we can always assume that we are working with independent features. This process is typically referred to as *whitening* and since this is not the primary topic of this chapter, we refer the reader to Fukunaga's book for an excellent treatise on the topic [22, pp.31-35].

When the number of classes is more than two, the question of dependence is extremely pertinent. Notice now that although the vectors can be transformed to permit a whitening process between any two classes, such a scheme will not be applicable for the rest of the classes. Thus the Bayes' classifier will have to resort to computing the Mahalanobis' distance in the original feature space.

The question is now one of modeling the dependence between features. In the strictest sense, it is possible that every feature is dependent on every other one. Thus, in general, both the covariance matrix Σ and its inverse Σ^{-1} are full. However

if the dependence is of the tree-type (i.e. x_i is dependent on x_j only if x_j is the parent of x_i) it is interesting to note that although Σ is full, Σ^{-1} is sparse¹ and fully captures the dependence information[12]. Since it is *this* quantity which is required in computing the Mahalanobis' distance, a significant gain in computing can be achieved in all such cases by modeling the dependence as a tree-type dependence.

In this chapter, we shall consider the problem of studying the dependence of an arbitrary random vector. To pose the problem in all its generality, we will not pay attention to the number of classes, but merely concentrate on a single random vector $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$ whose covariance matrix is Σ . In the first case we assume that there is a tree-type dependence between the components of $\mathbf{X}$. We shall study the problem of estimating this dependence tree and show that the estimate of the dependence tree converges to the true underlying (unknown) tree. However, if the underlying dependence is not of the tree type, our endeavor is to obtain the best tree-type dependence which adequately models Σ .

At this juncture, it is appropriate to query the word "best". Indeed, by this we mean that we are investigating how one can quantify the "goodness" of any tree. In this context, the most straightforward metric is the well-known Information Theoretic comparison between the "real" distribution and the distribution based on a given tree dependence. The computation of the best dependence tree naturally leads to the Expected Mutual Information Measure (EMIM) metric for capturing the dependence between *pairs* of variables. Although this metric is exact, it is difficult to compute, because of the logarithms, exponentials, and ratios involved.

In contrast to this, as in Chapter 2 we advocate the use of an alternate metric, the chi-squared metric, to represent dependence information between pairs of random variables. The latter involves no logarithms but is merely computed using sum of squares. In the case of discrete valued features, in Chapter 2 we showed analytically that for various families of distributions, the chi-squared metric **always** yields the optimal tree. In other cases, the experimentally obtained accuracy of it yielding the optimal tree was 100%. However, in this chapter we shall show that for the case of

---

¹This is a fascinating property. We shall justify it in Section 3.1.2.

continuous normal vectors, the result is much more powerful. Amazingly enough, we show that the chi-squared metric **always** yields the optimal tree, implying that the entire process of computing and manipulating the EMIM metric can be short-circuited.

Apart from the above results, we shall also prove results that involve the estimate of Σ (denoted as $\hat{\Sigma}$) as opposed to Σ itself. We shall show that the tree obtained by using the estimate $\hat{\Sigma}$ is the Maximum Likelihood Estimate (MLE) of the dependence tree, chosen from among all the possible trees, and that this estimate converges to the "real" unknown underlying tree as the number of samples increases. We note in passing, that in typical cases, this estimate converges to the true underlying tree when the number of samples is even as small as 50!!

3.1 Approximating Continuous Distributions

In the case when the vector $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$ is composed of continuous random variables $\{x_i\}$, $P(\mathbf{X})$ denotes the joint density function of the random vector $\mathbf{X}$. As in the discrete case, this joint distribution is said to be of *tree dependence* if $P(\mathbf{X})$ can be written in the form:

$$P(\mathbf{X}) = \prod_{i=1}^N P(x_{m_i} | x_{m_{j(i)}}) \quad (3.1)$$

where $0 \leq j(i) < i$ and $\langle m_1, m_2, \dots, m_N \rangle$ is a permutation of the integers in the set $\{1, 2, \dots, N\}$ and $m_0 = 0$.

Observe that in the RHS of (3.1), the term $P(x_{m_i} | x_{m_{j(i)}})$ represents the conditional density of the random variable x_{m_i} , conditioned on the random variable $x_{m_{j(i)}}$. Equation (3.1) can be seen to be a straightforward extension of (2.3) for the discrete vector case, obtained by substituting probability density functions in place of absolute probability masses. Observe too, that in the dependence tree τ corresponding to the approximation (3.1), there is an edge between the nodes x_{m_i} and $x_{m_{j(i)}}$. (3.1) can be easily rewritten in the symmetric form as follows:

$$P(\mathbf{X}) = \prod_{i=1}^N \frac{P(x_{m_i}, x_{m_{j(i)}})}{P(x_{m_i})P(x_{m_{j(i)}})} \prod_{i=1}^N P(x_{m_i})$$

$$= \prod_{[i,j] \in \tau} \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \prod_{i=1}^N P(x_i) \quad (\text{since } \langle m_1, m_2, \dots, m_N \rangle \text{ is a permutation}) \quad (3.2)$$

Since the notion of tree dependence in continuous distributions has been established, we now proceed to determine the best dependence tree² for a given continuous distribution.

3.1.1 Optimum Dependence Tree

In selecting the best approximating density function, we use as the measure of closeness of approximation, the natural extension to the information theoretic measure defined in (1.2). The metric for closeness between the “real” density function $P(\mathbf{X})$ and the approximating density $P_a(\mathbf{X})$ is quite simply given by:

$$I(P, P_a) = \int P(\mathbf{X}) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} d\mathbf{X}. \quad (3.3)$$

Note that the above integral is an N -fold *vector* integral with respect to the variables $x_1, x_2, \dots, x_N$. Again, as in the discrete case, it is well known that $I(P, P_a) \geq 0$ with the equality holding only if $P(\mathbf{X}) = P_a(\mathbf{X})$ almost everywhere.

In the case when the tree dependence is being used as model for approximation, the probability density derived from the dependence tree replaces the term $P_a(\mathbf{X})$ in (3.3). If the dependence tree in question is denoted by τ , the following equation gives the measure of closeness for the approximate density function derived from τ [12]:

$$I(P, P_\tau) = K - \sum_{[i,j] \in \tau} I^*(x_i, x_j) \quad (3.4)$$

where K is a constant independent of the approximation, and

---

²If the dependence tree is of the tree-type, the best dependence tree will be the underlying tree — and this is merely a consequence of the *metric* properties of the closeness measure. In all other cases, the best dependence tree τ is the one which minimizes the closeness measure between the “real” density function $P(\mathbf{X})$ and the density $P_\tau(\mathbf{X})$ based on the tree dependence model with the tree being τ .

$$I^*(x_i, x_j) = E \left[\log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \right] \quad (3.5)$$

Notice that the quantity $I^*(x_i, x_j)$ is simply the EMIM between the variables x_i and x_j . An observation of (3.4) shows that its LHS can be minimized by choosing the dependence tree to be the Maximum Spanning Tree (MST) of the complete graph with the EMIM values as the edge weights. This result is quite analogous to the discrete variable case.

There is one important difference to note here. In the discrete case, the evaluation of the EMIM between two variables x_i and x_j only depends on actual discrete probabilities of the joint events. In the continuous case, we encounter probability densities instead of actual discrete probabilities, and hence, there is no straightforward method to numerically compute the quantity defined by (3.5) if the analytic form of the density function is not known. Even if the analytic form of the densities are known, the quantity defined by (3.5) may not be computable, if the integrals cannot be evaluated to closed-form expressions³. We now investigate the best dependence tree for the case when the random vector $\mathbf{X}$ is normally distributed.

3.1.2 Optimal Dependence Tree for Normal Distributions

We assume that the random vector $\mathbf{X} = [x_1, x_2, \dots, x_N]^T$ is jointly normal with mean vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_N]^T$ and the covariance matrix $\Sigma = [\sigma_{ij}]$. The distribution of $\mathbf{X}$ will be denoted as $N(\boldsymbol{\mu}, \Sigma)$ and the density function $P(\mathbf{X})$ is given by:

$$P(\mathbf{X}) = (2\pi)^{-N/2} |\Sigma|^{-0.5} \exp \left\{ -\frac{1}{2} (\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \right\}, \quad (3.6)$$

where $|\Sigma|$ is the determinant and Σ^{-1} is the inverse of the covariance matrix Σ . Notice that Σ is an $N \times N$ symmetric, positive definite matrix and hence its inverse exists.

The marginal density of the random variable x_i has the usual uni-variate normal density $P(x_i)$ given by:

---

³For example, for the Cauchy distributions, the mean is not defined.

$$P(x_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp \left\{ -\frac{1}{2} \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 \right\}, \quad (3.7)$$

where σ_i is the square root of the diagonal entry σ_{ii} of the covariance matrix Σ . Also the joint distribution of the pair of random variables x_i and x_j has the density $P(x_i, x_j)$ given by:

$$P(x_i, x_j) = \frac{1}{2\pi\sigma_i\sigma_j(1 - \rho_{ij}^2)^{1/2}} \exp\{-q(x_i, x_j)\} \quad (3.8)$$

where,

$$\begin{aligned} q(x_i, x_j) &= \frac{1}{2(1 - \rho_{ij}^2)} \left[\left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 - 2\rho_{ij} \left(\frac{x_i - \mu_i}{\sigma_i} \right) \left(\frac{x_j - \mu_j}{\sigma_j} \right) \right] \end{aligned} \quad (3.9)$$

and $\rho_{ij} = \sigma_{ij}/\{\sigma_i\sigma_j\}$ is the correlation coefficient between the two variables x_i, x_j .

The reader must observe that (3.8) can be obtained from the general form (3.6) by taking the mean vector μ and Covariance matrix Σ to be column vector of dimension 2×1 and a square matrix of size 2×2 respectively. The entries in these matrices involve only parameters related to the variables x_i and x_j . This can alternatively be obtained by integrating the total density function to get the pairwise joint density function.

In order to motivate the algorithm to compute the optimum dependence tree, we now state the following result.

Lemma 3.1 *For all jointly normal pairs of random variables x_i, x_j , the EMIM metric between these variables is given by:*

$$E \left[\log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \right] = -\frac{1}{2} \log(1 - \rho_{ij}^2).$$

Proof:

The ratio of the joint and marginal densities can be obtained by combining (3.7) and (3.8) as follows:

$$\frac{P(x_i, x_j)}{P(x_i)P(x_j)}$$

$$= (1 - \rho_{ij}^2)^{-1/2} \exp \left[\frac{\rho_{ij}}{1 - \rho_{ij}^2} \left(\frac{x_i - \mu_i}{\sigma_i} \right) \left(\frac{x_j - \mu_j}{\sigma_j} \right) - \frac{\rho_{ij}^2}{2(1 - \rho_{ij}^2)} \left\{ \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right\} \right] \quad (3.10)$$

and hence,

$$\log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} = -\frac{1}{2} \log(1 - \rho_{ij}^2) + \left[\frac{\rho_{ij}}{1 - \rho_{ij}^2} \left(\frac{x_i - \mu_i}{\sigma_i} \right) \left(\frac{x_j - \mu_j}{\sigma_j} \right) - \frac{\rho_{ij}^2}{2(1 - \rho_{ij}^2)} \left\{ \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right\} \right] \quad (3.11)$$

Taking expectations of both sides of (3.11), we get,

$$\begin{aligned} E \left[\log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \right] \\ = -\frac{1}{2} \log(1 - \rho_{ij}^2) + \left[\frac{\rho_{ij}}{1 - \rho_{ij}^2} E \left[\left(\frac{x_i - \mu_i}{\sigma_i} \right) \left(\frac{x_j - \mu_j}{\sigma_j} \right) \right] - \frac{\rho_{ij}^2}{2(1 - \rho_{ij}^2)} \left\{ E \left[\left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 \right] + E \left[\left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right] \right\} \right] \end{aligned} \quad (3.12)$$

The RHS of (3.12) can be simplified making use of the fact that $E[(x_i - \mu_i)(x_j - \mu_j)] = \sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$ and that $E[(x_i - \mu_i)^2/\sigma_i^2] = 1$. This leads us to:

$$E \left[\log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \right] = -\frac{1}{2} \log(1 - \rho_{ij}^2). \quad (3.13)$$

and the lemma is proved. $\square$

As a result of Lemma 3.1, we can now algorithmically present the procedure to compute the best dependence tree for jointly normal random variables. This is given formally⁴ as Algorithm DepTree_EMIM in Program 3.1.

---

⁴It is interesting that although the random variables may have "functionally dependent" means, the mean vector itself does not come into the picture when evaluating stochastic dependence.

Algorithm DepTree_EMIM(μ, Σ, τ^*)**Input:**

The parameters μ and Σ for the normal distribution obeyed by $\mathbf{X}$.

Output:

The best dependence tree, τ^* , as per the EMIM metric.

Method:

1. From Σ , compute $R := [\rho_{ij}]$, the matrix of correlation coefficients.
2. Form a complete undirected weighted graph $G^* = (V, E, W^*)$, where

$$V := \{x_1, x_2, \dots, x_N\}$$

$$E := \{\langle x_i, x_j \rangle | i, j = 1, 2, \dots, N; i \neq j\}$$

$$W^*(\langle x_i, x_j \rangle) := I^*(x_i, x_j) = -\frac{1}{2} \log(1 - \rho_{ij}^2).$$
3. $\tau^* := \text{MaximumSpanningTree}(G^*)$

End Algorithm DepTree_EMIM

Program 3.1: Algorithm to Determine the Optimal Dependence Tree for Normal Distributions.

Having described the procedure to compute the best dependence tree for Gaussian Distributions, it can be beneficial to try to visualize the distribution characterized by *tree dependence*. Consider (3.2) which determines the form of the approximating density function generated by the dependence tree. Notice that the term $\prod_{i=1}^N P(x_i)$ is a product of densities of the form defined by (3.7) and leads to expressions which possess purely quadratic terms in their exponents. Furthermore, the first product in (3.2) groups terms whose form is given by (3.10). The net result is a density function with a form exactly identical to a normally distributed variable. In other words, the tree dependence for jointly normal vectors, approximates $N(\mu, \Sigma)$ by another normal distribution $N(\mu, \Sigma_\tau)$. Whereas the means of the two are identical,⁵ the covariance matrix of the latter is a "condensed" form of the covariance matrix of the former.

⁵Although this is not obvious, it becomes clear when one studies the quadratic form in the exponent of the exponential.

As a consequence of the above, it can be seen that the “cross-product” terms of the form $(x_i - \mu_i)(x_j - \mu_j)/\{\sigma_i\sigma_j\}$ appear only for those indices for which there is an edge between nodes x_i and x_j in the dependence tree. The number of such “cross-terms” is clearly $N - 1$, which is the number of branches in the trees. Combining this information with the fact that such cross terms are contributed to by the off-diagonal terms of the symmetric matrix Σ_τ^{-1} , we can conclude that although Σ_τ may be full, Σ_τ^{-1} is sparse and has at most $3N - 2$ elements, which are the N diagonal and $2(N - 1)$ off-diagonal elements !!

The main strength of using the tree approximation is that the computation of Σ_τ^{-1} can be achieved without the need to invert Σ_τ . This is a natural consequence of the fact that the elements of this square matrix can be simply expressed in terms of the relevant means, the standard deviations, and the correlation coefficients. The applications of approximating normal distributions with tree dependence has been presented in [12]. We now investigate the dependence tree that results if the chi-squared metric is used for the edge weights.

It must be emphasized that Algorithm DepTree_EMIM assumes that the “true” covariance matrix of the distribution of the random vector $\mathbf{X}$ is known. On the other hand, if only a finite number of samples of the vector $\mathbf{X}$ are available, the question is now one of obtaining a MLE (or for that matter any consistent estimate) of the dependence tree. We shall tackle this matter in a later section and turn our attention to the chi-squared metric being proposed in this chapter.

3.1.3 χ^2 Metric For Determining Dependence Trees

Let us now turn our attention to the problem of using a more computationally effective metric for determining the dependence tree. This metric is a generalization of the chi-squared metric, $I_\chi(x_i, x_j)$, introduced in Chapter 2. For continuous random variables this quantifies the degree of dependence between random variables x_i, x_j as

$$I_\chi(x_i, x_j) = \iint \frac{\{P(x_i, x_j) - P(x_i)P(x_j)\}^2}{P(x_i)P(x_j)} dx_i dx_j \quad (3.14)$$

From the definition of $I_\lambda(x_i, x_j)$, it is clear that I_λ has the following desirable properties that any metric which captures dependence information should possess:

1. $I_\lambda(x_i, x_j) \geq 0$
2. $I_\lambda(x_i, x_j) = 0$ if $P(x_i, x_j) = P(x_i)P(x_j)$

In view of the fact that the I^* metric appears naturally in the process of selecting the best dependence tree for continuous distributions, it might be inferred that I_λ cannot possibly lead to the optimal dependence tree. We shall prove the counter-intuitive result that for Gaussian distributions I_λ can be used instead of I^* to yield the *optimal* dependence tree. In other words, this increase in computational efficiency does not cause us to forfeit any optimality.

To do this, we first state our next result which involves the expression for $I_\lambda(x_i, x_j)$ for the case of the normal random vector $\mathbf{X}$. In the interest of brevity, many of the intermediate steps in the derivations are omitted.

Theorem 3.1 *If $\mathbf{X}$ has a Gaussian distribution $N(\mu, \Sigma)$, then $I_\lambda(x_i, x_j) = \frac{\rho_{ij}^2}{1 - \rho_{ij}^2}$.*

Proof:

$I_\lambda(x_i, x_j)$ given by (3.14) can be rewritten as:

$$I_\lambda(x_i, x_j) = \int \int \frac{(P(x_i, x_j))^2}{P(x_i)P(x_j)} dx_i dx_j - 1.$$

Now, from (3.8) we can easily write down the expression for $(P(x_i, x_j))^2$ (denoted for convenience as $P^2(x_i, x_j)$) as:

$$P^2(x_i, x_j) = \frac{1}{4\pi^2 \sigma_i^2 \sigma_j^2 (1 - \rho_{ij}^2)} \exp \{-2q(x_i, x_j)\} \quad (3.15)$$

where, $q(x_i, x_j)$ is given by (3.9).

Combining (3.15) and (3.7), we get,

$$\frac{P^2(x_i, x_j)}{P(x_i)P(x_j)} = \frac{1}{2\pi \sigma_i \sigma_j (1 - \rho_{ij}^2)} \exp \{r(x_i, x_j)\}$$

where,

$$r(x_i, x_j) = - \left\{ \left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right\} \frac{1 + \rho_{ij}^2}{2(1 - \rho_{ij}^2)} + \frac{2\rho_{ij}}{1 - \rho_{ij}^2} \left(\frac{x_i - \mu_i}{\sigma_i} \cdot \frac{x_j - \mu_j}{\sigma_j} \right).$$

Consider the integral:

$$I' = \iint \frac{P^2(x_i, x_j)}{P(x_i)P(x_j)} dx_i dx_j.$$

By making the simple substitution of variables

$$\xi = \frac{x_i - \mu_i}{\sigma_i} \frac{1}{\sqrt{1 - \rho_{ij}^2}}, \quad \text{and,} \quad \eta = \frac{x_j - \mu_j}{\sigma_j} \frac{1}{\sqrt{1 - \rho_{ij}^2}}$$

the integral I' can be seen to simplify to:

$$I' = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp \left\{ \left(-\frac{1 + \rho_{ij}^2}{2} \right) \cdot \xi^2 \right\} \int_{-\infty}^{\infty} \exp \left\{ \left(-\frac{1 + \rho_{ij}^2}{2} \right) \left(\eta^2 - \frac{4\xi\rho_{ij}\eta}{1 + \rho_{ij}^2} \right) \right\} d\xi d\eta.$$

Observe that the vector integral can now be evaluated by first evaluating the integral w.r.t. the variable η and then performing the integration w.r.t. variable ξ . The integration w.r.t. η can be completed, if we observe that the exponent of the exponential can be reduced to a purely quadratic term by “completing the square”. Subsequently, we can make use of the standard definite integral given below:

$$\int_{-\infty}^{+\infty} \exp \{ -a^2 x^2 \} dx = \frac{\sqrt{\pi}}{a}.$$

By carrying out the steps mentioned above, the integral w.r.t. the variable η can be shown to be:

$$\int_{-\infty}^{\infty} \exp \left\{ \left(-\frac{1 + \rho_{ij}^2}{2} \right) \left(\eta^2 - \frac{4\xi\rho_{ij}\eta}{1 + \rho_{ij}^2} \right) \right\} d\eta = \exp \left\{ \frac{2\rho_{ij}^2 \xi^2}{1 + \rho_{ij}^2} \right\} \cdot \sqrt{\frac{2\pi}{1 + \rho_{ij}^2}}$$

Therefore, the integral I' can be simplified to:

$$\begin{aligned}
 I' &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp \left[\left\{ -\frac{1 + \rho_{ij}^2}{2} + \frac{2\rho_{ij}^2}{1 + \rho_{ij}^2} \right\} \xi^2 \right] d\xi \cdot \sqrt{\frac{2\pi}{1 + \rho_{ij}^2}} \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp \left[\left\{ -\frac{(1 - \rho_{ij}^2)^2}{2(1 + \rho_{ij}^2)} \right\} \xi^2 \right] d\xi \cdot \sqrt{\frac{2\pi}{1 + \rho_{ij}^2}} \\
 &= \frac{1}{1 - \rho_{ij}^2}
 \end{aligned}$$

Hence the chi-squared metric between the two variables x_i and x_j has the form:

$$I_X(x_i, x_j) = I' - 1 = \frac{\rho_{ij}^2}{1 - \rho_{ij}^2}. \quad (3.16)$$

and the theorem is proved. $\square$

Thus far, we have shown how to compute the edge weights $I_X(x_i, x_j)$ in terms of the correlation coefficient ρ_{ij} . It is now a trivial matter to incorporate these edge weights to yield a dependence tree. The procedure to do this is given below in Algorithm DepTree_CHI⁶ (see Program 3.2).

---

⁶Please see footnote for the input parameters in Algorithm DepTree_EMIM.

Algorithm DepTree.CHI (μ, Σ, τ_x)**Input:**

The parameters μ and Σ for the normal distribution obeyed by $\mathbf{X}$.

Output:

The dependence tree, τ_x , as per the chi-squared metric.

Method:

1. From Σ , compute $R := [\rho_{ij}]$, the matrix of correlation coefficients.
2. Form a complete undirected weighted graph $G_x = (V, E, W_x)$, where

$$V := \{x_1, x_2, \dots, x_N\}$$

$$E := \{(x_i, x_j) | i, j = 1, 2, \dots, N; i \neq j\}$$

$$W_x((x_i, x_j)) := I_x(x_i, x_j) = \frac{\rho_{ij}^2}{1 - \rho_{ij}^2}.$$
3. $\tau_x := \text{MaximumSpanningTree}(G_x)$

End Algorithm DepTree.CHI

Program 3.2: Algorithm to Determine the Dependence Tree for Normal Distributions as per the chi-squared metric.

Having given a procedure to obtain the dependence tree τ_x , the question of how τ_x relates to the optimal tree τ^* has now to be considered. Indeed, we shall show that τ_x is exactly identical to τ^* , the best dependence tree. To establish this claim, we first state an auxiliary result.

Theorem 3.2 *Let x_i and x_j be any two components of the normal random vector $\mathbf{X}$. Then $I_x(x_i, x_j)$ is a monotonic function of $I^*(x_i, x_j)$.*

Proof:

Equation (3.13) which gave the expression for $I^*(x_i, x_j)$ can be rewritten as:

$$\rho_{ij}^2 = 1 - \exp\{-2I^*(x_i, x_j)\} \quad (3.17)$$

Substituting the expression for ρ_{ij}^2 into (3.16), we get

$$\begin{aligned} I_x(x_i, x_j) &= \frac{1 - \exp\{-2I^*(x_i, x_j)\}}{\exp\{-2I^*(x_i, x_j)\}} \\ &= \exp\{2I^*(x_i, x_j)\} - 1 \end{aligned} \quad (3.18)$$

whence the claim that $I_x(x_i, x_j)$ is a monotonic function of $I^*(x_i, x_j)$ follows. $\square$

This theorem naturally leads us to our fundamental result which establishes that τ_x and τ^* are the same.

Theorem 3.3 *Let $\mathbf{X}$ be a normally distributed random vector. Then τ_x (obtained by Algorithm DepTree_CHI) is either identical to, or is an equally good approximation as the tree τ^* produced by the Algorithm DepTree_EMIM.*

Proof:

Consider the complete graphs G^* and G_x . By virtue of (3.18), we can arrive at the straightforward conclusion that

$$W^*(x_i, x_j) > W^*(x_k, x_l) \Rightarrow W_x(x_i, x_j) > W_x(x_k, x_l) \quad \forall i \neq j, \quad k \neq l.$$

This implies that the relative ordering among the $W^*(\cdot, \cdot)$ and the $W_x(\cdot, \cdot)$ weights are identical.

The equivalence of the tree produced by the I^* and the I_x metrics will be shown by simulating Kruskal's algorithm [2]. This algorithm essentially sorts the edges in the descending order of weights and selects the $(N - 1)$ edges that do not form cycles. It is apparent that since the relative ordering of the edge weights under the two metrics are identical, the corresponding MSTs computed will be identical.

It must be noted however that the MST is unique only if all the edge weights are distinct. If two edges have identical weights under I^* metric, it can be trivially observed from (3.18) that they have identical weights under the I_x metric. Thus in these cases, the MST's produced by the two metrics can only differ in the selection of an edge from the set of edges with equal weight. The sum of the weights of the edges included in the MST will indeed be identical and hence both the MSTs will be equally good approximations. Hence the theorem. $\square$

We shall now turn our attention to the more fundamental task of estimating the dependence tree from the set of samples $\mathcal{X}$ instead of the parameters μ and Σ .

3.2 Estimating Dependence Tree From Samples

Throughout our presentation, we implicitly worked with the assumption that the “true” covariance matrix of the random vector $\mathbf{X}$ was available to us. This knowledge enabled us to determine the optimal dependence tree by Algorithm `DepTree_EMIM`, or equivalently, by the computationally efficient Algorithm `DepTree_CHI`. The situation however changes if instead of the true parameters of the normal distribution of the random vector $\mathbf{X}$, only a finite number of representative samples are presented to the procedure which estimates the dependence tree.

An obvious method can be envisioned. Since the covariance matrix is unknown, one can first obtain its MLE and subsequently follow either the procedure of Algorithm `DepTree_EMIM` or `DepTree_CHI`. This procedure is intuitively appealing, but its properties as an estimator of the *dependence tree* are not obvious. We shall now prove that this simple minded and straightforward procedure indeed produces the MLE of the dependence tree when the MLEs of the corresponding parameters $\hat{\mu}$ and $\hat{\Sigma}$ are used in Algorithm `DepTree_CHI`.

Consider Algorithm `Estimate_DepTree_CHI` (in Program 3.3). The reader should observe that this algorithm is identical to Algorithm `DepTree_CHI` except that instead of having the input parameters as μ and Σ , the input is a set $\mathcal{X}$ of s statistically independent observations of $\mathbf{X}$.

Algorithm Estimate_DepTree_CHI ($\mathcal{X}, \hat{\tau}$)**Input:** $\mathcal{X} = s$ statistically independent samples $X^1, X^2, \dots, X^s$.**Output:**An estimate of the dependence tree, $\hat{\tau}$, as per the chi-squared metric.**Method:**

1. Compute the MLEs of the parameters of the distribution of $\mathbf{X}$.

$$\hat{\boldsymbol{\mu}} := \frac{1}{s} \sum_{k=1}^s X^k, \quad \text{and,}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{s} \sum_{k=1}^s (X^k - \hat{\boldsymbol{\mu}})(X^k - \hat{\boldsymbol{\mu}})^T$$

2. Invoke Algorithm DepTree_CHI ($\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}, \hat{\tau}$)

End Algorithm Estimate_DepTree_EMIM**Program 3.3:** Estimate Dependence Tree from Samples.

We now state and prove the fundamental property of the tree estimated by Algorithm Estimate_DepTree_CHI.

Theorem 3.4 *Algorithm Estimate_DepTree_CHI produces the Maximum Likelihood Estimate of the dependence tree.*

Proof:

First of all note that a tree type approximation to the given density function is completely described by the structure of the tree and the parameters associated with this tree. The parameters to be chosen consist of the Mean Vector, the Variances of all the components of the random vector and the correlation coefficient for the pair of nodes connected by an edge in the dependence tree. In the following derivation, we shall denote the set of parameters as $\boldsymbol{\theta}$ and the tree by τ .

Let $X^1, X^2, \dots, X^s$ be the s statistically independent samples drawn from the underlying normal distribution. Let $p(X^k | \boldsymbol{\theta}, \tau)$ denote the value of the density function for the sample X^k , given the set of current parameter values and the dependence tree τ . Then, using (3.2), we can write:

$$p(X^k|\theta, \tau) = \prod_{[i,j] \in \tau} \frac{P(x_i^k, x_j^k)}{P(x_i^k)P(x_j^k)} \cdot \prod_{l=1}^N P(x_l^k),$$

where x_j^k is the j^{th} component of the sample X^k . Equivalently, by taking logarithms,

$$\log p(X^k|\theta, \tau) = \sum_{[i,j] \in \tau} \log \frac{P(x_i^k, x_j^k)}{P(x_i^k)P(x_j^k)} + \sum_{l=1}^N \log P(x_l^k).$$

Utilizing (3.11), we obtain:

$$\log \frac{P(x_i^k, x_j^k)}{P(x_i^k)P(x_j^k)} = -\frac{1}{2} \log(1 - \rho_{ij}^2) - R(x_i^k, x_j^k)$$

where,

$$R(x_i^k, x_j^k) = - \left[\frac{\rho_{ij}}{1 - \rho_{ij}^2} \left(\frac{x_i^k - \mu_i}{\sigma_i} \right) \left(\frac{x_j^k - \mu_j}{\sigma_j} \right) - \frac{\rho_{ij}^2}{2(1 - \rho_{ij}^2)} \left\{ \left(\frac{x_i^k - \mu_i}{\sigma_i} \right)^2 + \left(\frac{x_j^k - \mu_j}{\sigma_j} \right)^2 \right\} \right]$$

Also, utilizing the expression for the univariate Gaussian density given by (3.7), we can obtain the log-likelihood function ℓ as:

$$\begin{aligned} \ell &= \sum_{k=1}^s \log p(X^k|\theta, \tau) \\ &= \sum_{k=1}^s \left[\sum_{[i,j] \in \tau} \left\{ -\frac{1}{2} \log(1 - \rho_{ij}^2) - R(x_i^k, x_j^k) \right\} \right. \\ &\quad \left. - \sum_{l=1}^N \left\{ \frac{1}{2} \left(\frac{x_l^k - \mu_l}{\sigma_l} \right)^2 + \frac{1}{2} \log(2\pi) + \log \sigma_l \right\} \right] \\ &= \sum_{[i,j] \in \tau} \left[-\frac{s}{2} \log(1 - \rho_{ij}^2) - \sum_{k=1}^s R(x_i^k, x_j^k) \right] \\ &\quad - \sum_{l=1}^N \sum_{k=1}^s \left\{ \frac{1}{2} \left(\frac{x_l^k - \mu_l}{\sigma_l} \right)^2 + \frac{1}{2} \log(2\pi) + \log \sigma_l \right\}. \end{aligned} \quad (3.19)$$

In trying to arrive at the best approximation, we are faced with a complex optimization problem, which is that of choosing the dependence tree, and of choosing its associated parameters. The problem would be in a sense "decomposable" if we can independently choose the tree and the values of the parameters. Fortunately we can show that the "best" values of the parameters, $\hat{\theta}$, are the same, irrespective

of what the estimated dependence tree is. This, in turn, permits us to perform the maximization of ℓ in a straightforward fashion.

We shall first investigate the values of the parameters in θ that would maximize the quantity ℓ for a *given tree*. In order to achieve this, we merely have to differentiate ℓ w.r.t. θ and set the result to 0, which in turn would yield the estimate $\hat{\theta}$. The set of equations,

$$\frac{\partial \ell}{\partial \mu_\alpha} = 0 \quad \text{for } \alpha = 1, 2, \dots, N$$

simplifies to

$$-\sum_{[\alpha, j] \in \tau} \sum_{k=1}^s \frac{\partial R(x_\alpha^k, x_j^k)}{\partial \mu_\alpha} + \sum_{k=1}^s \left(\frac{x_\alpha^k - \mu_\alpha}{\sigma_\alpha} \right) = 0,$$

which, after some manipulation, yields,

$$\begin{aligned} -\sum_{[\alpha, j] \in \tau} \frac{1}{2(1 - \rho_{\alpha j}^2)} \left[\left\{ \sum_{k=1}^s 2 \left(\frac{x_\alpha^k - \mu_\alpha}{\sigma_\alpha} \right) (-1) \right\} \rho_{\alpha j}^2 \right. \\ \left. - \sum_{k=1}^s 2 \rho_{\alpha j} \left(-\frac{1}{\sigma_\alpha} \right) \left(\frac{x_j^k - \mu_j}{\sigma_j} \right) \right] + \sum_{k=1}^s \left(\frac{x_\alpha^k - \mu_\alpha}{\sigma_\alpha} \right) = 0, \end{aligned}$$

which equivalently is:

$$\begin{aligned} \sum_{[\alpha, j] \in \tau} \frac{\rho_{\alpha j}^2}{(1 - \rho_{\alpha j}^2)} \cdot \left\{ \sum_{k=1}^s \left(\frac{x_\alpha^k - \mu_\alpha}{\sigma_\alpha} \right) \right\} - \frac{\rho_{\alpha j}}{(1 - \rho_{\alpha j}^2) \sigma_\alpha} \cdot \left\{ \sum_{k=1}^s \left(\frac{x_j^k - \mu_j}{\sigma_j} \right) \right\} \\ + \left\{ \sum_{k=1}^s \left(\frac{x_\alpha^k - \mu_\alpha}{\sigma_\alpha} \right) \right\} = 0 \end{aligned} \quad (3.20)$$

Differentiating ℓ w.r.t. $\rho_{\alpha\beta}$, where $[\alpha, \beta] \in \tau$, we get:

$$\frac{\partial \ell}{\partial \rho_{\alpha\beta}} = 0$$

which can be written down as ⁷:

$$-\frac{s(-2\rho_{\alpha\beta})}{2(1 - \rho_{\alpha\beta}^2)} - \sum_{k=1}^s \frac{\partial R(x_\alpha^k, x_\beta^k)}{\partial \rho_{\alpha\beta}} = 0,$$

---

⁷Notice that we have made use of the results that the derivatives of the functions $\rho/(1 - \rho^2)$ and $\rho^2/(1 - \rho^2)$ w.r.t. ρ are $(1 + \rho^2)/(1 - \rho^2)^2$ and $2\rho/(1 - \rho^2)^2$ respectively.

and can be further simplified to:

$$\begin{aligned} \frac{s\rho_{\alpha\beta}}{1-\rho_{\alpha\beta}^2} - \frac{\rho_{\alpha\beta}}{(1-\rho_{\alpha\beta}^2)^2} \left[\sum_{k=1}^s \left\{ \left(\frac{x_{\alpha}^k - \mu_{\alpha}}{\sigma_{\alpha}} \right)^2 + \left(\frac{x_{\beta}^k - \mu_{\beta}}{\sigma_{\beta}} \right)^2 \right\} \right] \\ + \frac{1+\rho_{\alpha\beta}^2}{(1-\rho_{\alpha\beta}^2)^2} \sum_{k=1}^s \frac{(x_{\alpha}^k - \mu_{\alpha})(x_{\beta}^k - \mu_{\beta})}{\sigma_{\alpha}\sigma_{\beta}} = 0 \end{aligned} \quad (3.21)$$

Also,

$$\frac{\partial \ell}{\partial \sigma_{\alpha}} = 0 \quad \text{for } \alpha = 1, 2, \dots, N,$$

yields (on simplification) the following equations:

$$\begin{aligned} - \sum_{[\alpha, j] \in \tau} \left[\frac{1}{1-\rho_{\alpha j}^2} \left\{ \rho_{\alpha j}^2 (-2) \sum_{k=1}^s \frac{(x_{\alpha}^k - \mu_{\alpha})^2}{\sigma_{\alpha}^3} \right\} + 2\rho_{\alpha j} \sum_{k=1}^s \frac{(x_{\alpha}^k - \mu_{\alpha})(x_j^k - \mu_j)}{\sigma_{\alpha}^2 \sigma_j} \right] \\ + \sum_{k=1}^s \frac{(x_{\alpha}^k - \mu_{\alpha})^2}{\sigma_{\alpha}^3} - \frac{s}{\sigma_{\alpha}} = 0. \end{aligned} \quad (3.22)$$

By direct substitution we can verify that (3.20), (3.21) and (3.22) are satisfied if:

$$\begin{aligned} \hat{\mu}_i &= \frac{1}{s} \sum_{k=1}^s x_i^k \\ \hat{\sigma}_i &= \frac{1}{s} \sum_{k=1}^s (x_i^k - \hat{\mu}_i)^2 \\ \hat{\rho}_{ij} &= \frac{1}{s\hat{\sigma}_i\hat{\sigma}_j} \sum_{k=1}^s (x_i^k - \hat{\mu}_i)(x_j^k - \hat{\mu}_j). \end{aligned}$$

which interestingly enough are the MLEs of the parameters of the distribution of $\mathbf{X}$.

Notice that all these parameters are **independent** of the tree τ . With these values of the parameters θ , let us consider the quantity $\sum_{k=1}^s R(x_i^k, x_j^k)$ which appears in (3.19). Now,

$$\begin{aligned} \sum_{k=1}^s R(x_i^k, x_j^k) \\ = \frac{\hat{\rho}_{ij}^2}{2(1-\hat{\rho}_{ij}^2)} \left[\sum_{k=1}^s \left\{ \left(\frac{x_i^k - \hat{\mu}_i}{\hat{\sigma}_i} \right)^2 + \left(\frac{x_j^k - \hat{\mu}_j}{\hat{\sigma}_j} \right)^2 \right\} \right] \\ - \frac{\hat{\rho}_{ij}}{1-\hat{\rho}_{ij}^2} \sum_{k=1}^s \frac{(x_i^k - \hat{\mu}_i)(x_j^k - \hat{\mu}_j)}{\hat{\sigma}_i\hat{\sigma}_j} \end{aligned}$$

$$\begin{aligned}
&= \frac{\hat{\rho}_{ij}^2}{2(1 - \hat{\rho}_{ij}^2)} \cdot 2s - \frac{\hat{\rho}_{ij}}{1 - \hat{\rho}_{ij}^2} \cdot \frac{s\hat{\sigma}_i\hat{\sigma}_j\hat{\rho}_{ij}}{\hat{\sigma}_i\hat{\sigma}_j} \\
&= 0.
\end{aligned}$$

And hence, ℓ in (3.19) can be simplified to:

$$\ell = -\frac{s}{2} \sum_{[i,j] \in \tau} \log(1 - \hat{\rho}_{ij}^2) + k'. \quad (3.23)$$

where k' is a constant independent of the tree and only depends only on the samples. An examination of (3.23) reveals that the MLE of the dependence tree can be found as the MST of a weighted graph, whose edge weights are the values of the corresponding EMIM metric, calculated using the *estimates* of the correlation coefficients. The theorem follows since we have already established that when presented with identical input, Algorithms DepTree_EMIM and DepTree_CHI *always* produce identical (or equivalent) trees. $\square$

Remark:

The reader will recall that the maximization of the log-likelihood function ℓ required the choice of the dependence tree, as well the parameters that completely specify the approximation. Also worth noting is the fact that the set of parameters associated with *different trees* are not identical. To clarify this, consider two dependence trees τ_1 , and τ_2 , where τ_1 includes the edge $\langle x_i, x_j \rangle$ and τ_2 does not. Clearly, the correlation coefficient, ρ_{ij} will be part of the parameter set associated with τ_1 . However, ρ_{ij} is not needed when approximating with the tree τ_2 . Viewed in this perspective, the proof presented above demonstrates that the parameters **common** to any two trees, must have the **same value**, in order to maximize the log-likelihood function. This simplifies the search process for the best approximation. Indeed, in every case, we merely choose all the means, variances and the correlation coefficients to be their respective MLEs and subsequently compute the best estimate of the dependence tree as the MST of the appropriately weighted graph.

3.3 Experimental Results

Simulations were carried out to validate the results in this chapter. Since the Chi-squared and the EMIM metric were analytically shown to be exactly equivalent for normal vectors, numerous experiments conducted to verify the optimality of the chi-squared metric are not reported here, because they do not add to the fundamental contribution of this chapter.

In our simulations, we chose to monitor the only characteristic that was not analytically studied — the rate of convergence of the algorithm. In these experiments, we chose an underlying dependence tree (denoted as τ_r) and observed how quickly the estimates τ_λ (and hence τ^*) converged to it. Besides observing the convergence to the real tree, we also observed the improvement in the generated approximation, as more and more samples were seen by the system attempting to learn the characteristics of the distribution. The simulations were carried out for various values of the dimension of the random vector $\mathbf{X}$. The following procedure was followed for each value of the dimension parameter N .

Once the dimension of the random vector $\mathbf{X}$ was assigned, a predefined dependence tree was selected as follows. The structure of the dependence tree was initially arbitrarily selected. The means and variances of each of the variables $\{x_i\}$ were then assigned to be independent, uniformly distributed values in the interval (0,1). Subsequently, for each edge $\langle x_i, x_j \rangle$ in the dependence tree, the correlation coefficient ρ_{ij} was uniformly chosen from the interval $(-1 \dots 1)$. All the parameters characterizing the tree dependence being specified, the inverse of the covariance matrix (i.e. Σ^{-1}) was computed. This matrix was inverted to obtain the “real” underlying covariance matrix Σ from which the samples, $\mathcal{A}$, were generated. These pre-processing steps were accomplished in order to completely describe the distribution of $\mathbf{X}$. The subsequent steps consisted of generating the samples and estimating the dependence tree from them.

Using the technique described in Appendix B samples were generated according

to the distribution $N(\boldsymbol{\mu}, \Sigma)$.⁸ We denote the k^{th} sample as $\mathbf{X}^k$. Let m_k and C_k denote the MLEs of the mean vector and the covariance matrix, after the k^{th} sample vector were seen. These estimates were updated as the samples were processed, using the recurrence relations[18, pp.82]:

$$\begin{aligned} m_{n+1} &= \frac{1}{n+1} [nm_n + \mathbf{X}^{n+1}] \\ C_{n+1} &= \frac{n}{n+1} C_n + \frac{n}{(n+1)^2} (\mathbf{X}^{n+1} - m_n)(\mathbf{X}^{n+1} - m_n)^T. \end{aligned}$$

where, $m_1 = \mathbf{X}^1$, and C_1 is the null matrix. Notice that the above recurrence relation for C_{n+1} is slightly different from the relation that appears in [18, pp.82]. This is due to the fact that C_n refers to the MLE of the covariance matrix, and is not the unbiased estimator used in [18, pp.82].

Once the estimated covariance matrix was available, Algorithm DepTree.CH1 was invoked to estimate the dependence tree. In our simulations, we opted not to compute the dependence tree for the first 50 samples so as to ensure that we were dealing with “reasonable” estimates for the mean vector and the covariance matrix.

It was observed during the simulations that the estimate $\hat{\tau}$ produced by Algorithm EstimateDepTree.EMIM converges very quickly to the underlying “real” dependence tree τ_r . However, the *distribution* generated by the estimate $\tau_{\hat{\tau}}$ converges more sluggishly to the underlying distribution, even though $\tau_{\hat{\tau}}$ and τ_r are structurally identical. This is essentially due to inaccurate estimates for the mean vector and the covariance matrix.

As far as experimental results are concerned, the gradual improvement in the “quality” of approximation, as the number of samples increases, is reported in the graphs given below. The quality of approximation, is of course the information theoretic closeness measure, $I(P, P_{\tau})$, between the “real” distribution $N(\boldsymbol{\mu}, \Sigma)$ and the approximating distribution $N(\hat{\boldsymbol{\mu}}, \hat{\Sigma})$. This closeness of approximation has the following explicit form:

$$I(P, P_{\tau}) = 0.5 \log \frac{|\hat{\Sigma}|}{|\Sigma|} - 0.5 \left\{ N - \text{tr}(\hat{\Sigma}^{-1}\Sigma) - (\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^T \hat{\Sigma}^{-1} (\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}) \right\}. \quad (3.24)$$

---

⁸We use the technique described in Fukunaga’s excellent book [22].

where $\text{tr}(A)$ denotes the trace of the square matrix A . That $I(P, P_7)$ has the form of (3.24) is shown in Appendix B.

We now present below the results obtained for two typical distributions.

In the first case, the dimension of the vector, N was 6. The underlying dependence tree is shown in Figure 3.1a. Figure 3.1b shows the closeness measure as a function of n , the number of samples. Note that the closeness measure reported in this figure is the ensemble average over 50 experiments, with each experiment lasting for 5000 samples. Observe the monotonic decrease in the closeness measure with the number of samples. Thus, for example, in Figure 3.1b, the closeness measure was 0.363854 after 50 samples were processed. This value dropped to 0.01 after processing 1000 samples. It was observed that the average closeness measure attained a value of 0.002681 after the 5000 samples were processed.

Figure 3.2 presents similar results for the case when the dimension of the vector, N , was 10. The experiments were conducted in a fashion identical to the one reported above. Again the initial value of the closeness measure was 0.996056, at the end of 50 samples. This value attains the value of 0.03220 and 0.0061, after 1000 and 5000 samples were processed, respectively. The power of the scheme is obvious.

(a) Underlying dependence tree used for generating the samples to measure the rate of convergence of τ_x . In this case, N , the dimensionality of the random vector is 6, which is equal to the number of nodes in this tree.

(b) This figure shows the ensemble average of the closeness of approximation $I(P, P_a)$ between the actual density P and the approximating density P_a derived from the estimated dependence tree. The underlying tree describing P is shown in Figure 3.1a.

Figure 3.1: Rate of Convergence. Continuous Vector case ($N=6$).

(a) Underlying dependence tree used for generating the samples to measure the rate of convergence of τ_λ . In this case, N , the dimensionality of the random vector is 10, which is equal to the number of nodes in this tree.

(b) This figure shows the ensemble average of the closeness of approximation $I(P, P_a)$ between the actual density P and the approximating density P_a derived from the estimated dependence tree. The underlying tree describing P is shown in Figure 3.2a.

Figure 3.2: Rate of Convergence. Continuous Vector case ($N=10$).

3.4 Summary

In this chapter, we have considered the problem of approximating the dependence information of a multi-variate normal distribution with one based on the structure of a dependence tree. The best dependence tree τ^* is known to be the MST of a complete graph, with $I^*(x_i, x_j)$ as the edge weight between the pair of nodes x_i and x_j , where I^* is the EMIM metric between the corresponding variables.

As in Chapter 2, this chapter proposes a chi-squared based metric, I_χ to capture the dependence information between pairs of random variables. For the case of normally distributed vectors, the I_χ metric is shown to be a monotonic function of the metric I^* . As a natural consequence of this property, τ_χ , the dependence tree found using the I_χ metric has been shown to be identical to, or as good an approximation as τ^* .

Instead of assuming that the actual parameters characterizing the normal distribution in question are available, the chapter also considers the situation in which only a finite number of samples of the vector are available. It is shown that the “natural” sequence of first estimating the parameters of the distribution, and then using these estimates *as if they were the true parameters*, yields the MLE of the dependence tree.

The chapter also includes simulations verifying and demonstrating the power of the strategy proposed. A summary of the results reported in this chapter have been published [92].

2

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

Chapter 4

Recognizing Sources Of Random Strings

The source recognition problem was briefly described in the introductory chapter in Section 1.1.2. Our aim is primarily to construct a supervised PR system which is capable of identifying the source which generated the random string presented to it. The two types of random string generators that are investigated in this chapter are the following. The first category of string generators generate random strings in which individual symbols can occur many times in the string. The second category deals primarily with sources which generate permutations of the chosen alphabet.

We first study the case of recognizing sources which generate arbitrary random strings.

4.1 The U-model for Arbitrary Strings

In Section 1.1.2 we alluded to the fact that a precise modeling of a source of random strings of length M , based on a finite alphabet $\mathcal{A}$ requires $|\mathcal{A}|^M$ estimates. In this section, we propose a very simple but useful model, called the unigram U-model, for such sources. This model uses $|\mathcal{A}|$ parameters instead of the previously indicated number.

In the U-model, U is a vector describing the distribution of the symbols of $\mathcal{A}$ in the received strings. In this model, it is assumed that each one of the M symbols is generated, in a manner independent of the prefix which has already been generated. The U-model essentially implies that the symbols of $\mathcal{A}$ are generated according to a

a multinomial distribution, U . Using this information, it is possible to compute the Maximum Likelihood Estimate (MLE) for the vector U for any specific source. It is easy to show that this MLE coincides with the intuitive frequency based estimate [18].

The U-model as described above simply corresponds to one in which only the unigrams of the symbols (i.e. individual probabilities of occurrence) are maintained. An immediate extension to this model can be suggested. In this, the U -vector need not remain fixed for all positions in a specific generated string, but could vary from position to position in a Markovian fashion. With some insight, this can be seen to exactly correspond to the bigram model of string generation.

We shall now explore the usefulness of the U-model in source recognition. It is clear that due to its simplicity this model *may* be an unreasonable one if the sources themselves use the bigram (or in general, the n -gram) probabilities for the generation strategy. In such cases, the U-model based on the *original* alphabet need not be accurate. However, it is conceivable that with regard to some transformed alphabet, a unigram model may be acceptable. To clarify this point consider the case when the sources generate 256 bytes in a single transmission unit, where the original alphabet may consist of the letters 'a' ... 'z' (or the ASCII set). Given the same block of 2048 bits, we can sample selected bit positions of this bit string, to form the primitive symbols of a *new* alphabet. Instead of modeling the unigrams of the symbols of the underlying source alphabet, it may be convenient and even efficient to model the source string using the unigrams of the *transformed* alphabet. This is indeed a justified assumption, as can be seen from the following sample application.

Consider the case when we have c sources $\Omega = \{\omega_1, \omega_2, \dots, \omega_c\}$ where each source is generating random strings according to the some underlying distribution D , *which need not be a unigram based distribution*. The output of each source is separately encrypted — for example, by using a common encryption scheme, but with each source having its own distinct “key”. This situation is depicted in Figure 4.1. Following the encryption, we effectively have a new set of c sources $\Omega' = \{\omega'_1, \omega'_2, \dots, \omega'_c\}$. Thus

distinguishing between these sources in Ω' , is equivalent to distinguishing between the original sources or alternatively, between the individual encryption keys.

To justify the use of the U-model in source recognition, we have conducted various experiments involving character text and a well known encryption scheme. In our experiments, the method used by the sources in the set Ω is the string generation technique based on English bigrams, and not on the unigrams alone. The encryption is performed using the Rivest-Shamir-Adleman (RSA) scheme¹[16]. In this scheme, if P and E denote the plain-text and the encrypted string respectively, the operation of encryption can be described as follows:

$$E = P^e \bmod (p * q)$$

where p, q , and e are parameters of the encryption scheme, and are referred to as the “key”. In this setting, the sources of multiple strings are simply obtained by encrypting the plain-text P under different “keys”. The problem of source recognition immediately leads to one of distinguishing between the keys used by the sources for their encryption.

The details about the experiments conducted are as follows. Table 4.1 gives a list of all the encryption keys used in the experiments². It must be noted that only a subset of these keys were used in any single experiment, and this subset depends on the number of sources being distinguished. Specifically, if the number of classes is c , we would use the keys $K_1, K_2, \dots, K_c$ from Table 4.1. In each case, the input data was generated as a continuous stream of characters, based on the English *bigrams*, and this stream was sub-divided into units of four bytes length. Each such sequence of the string generated by the source ω_i was encrypted using key K_i , and this stream of data was reckoned to be the encrypted bit string generated

---

¹The aim of this exercise is to use *any* encryption scheme in an effort to mask the information characteristic to the language. Thus we need not necessarily have used a public encryption scheme such as the RSA scheme whose primary advantage is its potential for being used in public key encryption. We request the reader to understand the spirit in which the encryption is being used.

²In real life situations p and q are many orders of magnitude larger than what is used in these experiments. We have just used representative values which satisfy the guidelines in [16, page 104].

by the source ω'_i . To obtain the relevant U-model, we have regarded four consecutive bits in this encrypted sample to represent one symbol of the transformed alphabet. Therefore, the alphabet describing the output of the sources in the set Ω' consists of 16 symbols, namely the hexadecimal digits $\{0 \dots 9, A \dots F\}$. Consequently, a unigram model for *this* output stream would comprise of a vector of dimension 16.

Figure 4.1: Set up for generating data for source recognition. Every source ω_i generates data based on the English bigram distribution. The data generated by the source ω_i is encrypted using the key K_i to yield the output of the pseudo-source ω'_i .

In our experiments, for each of the sources, the training phase presented the recognition system with 1000 sample encrypted bit strings. The features extracted from these samples was a single 16-dimensional vector for each source.

In order to test the accuracy of the recognition system, experiments were per-

Key	p	q	e
K_1	31	751	4901
K_3	43	409	13981
K_5	53	61	71
K_7	67	541	10493
K_9	73	127	5287
K_{11}	89	503	2247
K_{13}	101	281	20063
K_{15}	113	503	38133
K_{17}	127	151	15277
K_{19}	137	389	4817
K_{21}	149	179	25743
K_{23}	157	233	27305
K_{25}	167	229	32755
K_{27}	197	263	27341
K_{29}	211	257	51811
K_{31}	239	281	49369

Key	p	q	e
K_2	37	601	14059
K_4	47	631	24947
K_6	53	401	1461
K_8	71	349	23509
K_{10}	83	179	3145
K_{12}	97	443	19765
K_{14}	107	281	4351
K_{16}	113	257	23789
K_{18}	131	409	20209
K_{20}	139	317	14987
K_{22}	151	307	27467
K_{24}	163	307	41159
K_{26}	173	301	5413
K_{28}	199	263	46285
K_{30}	229	257	57397
K_{32}	541	31	211

Table 4.1: This table specifies the various keys used by the RSA scheme. A source ω_i is characterized by its unique key K_i . The results of source recognition are given in Table 4.2.

formed with varying number of classes. For each value of the number of classes, the recognition system was tested with 3200 different bit strings generated by the sources in the set Ω' . Each bit string Q_{test} was composed of only 100 4-byte units as described earlier. The system estimated the unigram vector for Q_{test} and subsequently utilized the Maximum Likelihood Classifier for determining the source of the string. For each value of the number of classes, the keys chosen for encryption are shown in Table 4.2. Observe that when the number of classes was 8, we obtained an accuracy of 99.93%. When the number of classes increased to 32, the recognition accuracy obtained was 99.56%. The results are indeed remarkable. What is perhaps more surprising is the fact that even though our classifier is not of the hierarchical type, it performs quite well in a 32 class problem.

From the above, we can see that even a simple model such as the U-model may be acceptable for source recognition if the underlying strings are arbitrary and are from a finite alphabet.

We now turn our attention to the far more complex and analytically interesting case when the strings generated are not arbitrary strings, but are constrained to be permutations. In this case, a model similar to the U-model is applicable, except that the associated pattern recognition phases of estimation and classification become more complicated because the dependence information is reckoned for due to the "with replacement" model of experimentation.

Experiment No.	No Of Classes	Keys Used	No of Missclassifications	Accuracy (in %)
I	8	$\{K_1 \dots K_8\}$	2	99.937
II	16	$\{K_1 \dots K_{16}\}$	3	99.906
III	24	$\{K_1 \dots K_{24}\}$	7	99.781
IV	32	$\{K_1 \dots K_{32}\}$	14	99.562

Table 4.2: This table shows the recognition accuracy obtained when the source recognition system was used to distinguish between several keys. For each experiment, 3200 bit strings were classified using the Maximum likelihood Classifier described in Section 4.1.

4.2 The S-Model for Permutation Generators

We now present our model for sources emitting random permutations of the entire alphabet $\mathcal{A} = \{A_1, A_2, \dots, A_M\}$. We use the model proposed by Oommen and Ng in [75] and assume that the underlying data generation strategy can be completely specified by a control vector of dimension M . This control vector, $\mathcal{S}$, is an $M \times 1$ probability vector denoted by $[s_1, s_2, \dots, s_M]^T$ which satisfies:

$$\sum_{i=1}^M s_i = 1.$$

Let $\mathcal{V}$ be any subset of $\mathcal{A}$. We define the vector $\mathcal{S}_{\mathcal{V}}$ as the **conditional** control vector of (normalized probabilities) in which only the quantities corresponding to the members of $\mathcal{V}$ have non-zero values. Thus $\mathcal{S}_{\mathcal{V}}$ consists of normalized probabilities $[s'_i]$, where:

$$s'_i = \begin{cases} 0 & \text{if } A_i \notin \mathcal{V} \\ \frac{s_i}{\sum_{A_i \in \mathcal{V}} s_i} & \text{otherwise} \end{cases} \quad (4.1)$$

Observe that the vector $\mathcal{S}$ is exactly equivalent to $\mathcal{S}_{\mathcal{A}}$.

Given the vector parameter $\mathcal{S}$, we assume that the underlying generation strategy is as follows. For the sake of explanation, let $X \in \Pi$ be a randomly generated piece of data (permutation) of $\mathcal{A}$. We denote X by $[x_1, x_2, \dots, x_M]$. The generation of a random permutation X clearly consists of randomly assigning a position to each symbol of $\mathcal{A}$. The S-model proposes that this is done by computing the prefixes of X in sequence. Initially x_1 is randomly assigned an symbol in $\mathcal{A}$, based on the control vector $\mathcal{S}$ (which is the same as $\mathcal{S}_{\mathcal{A}}$). By this we mean that symbol A_i is chosen with probability s_i . The subsequent problem is simply one of generating a (random) permutation of the $(M - 1)$ symbols of $\mathcal{A} - \{x_1\}$. The S-model proposes that this generation is done using the control vector $\mathcal{S}_{\mathcal{V}}$ and the process is recursively repeated by successively updating $\mathcal{V}$. For the sake of clarity, we present an algorithmic form of the above generation strategy.

Algorithm S-model**Input:**

The control vector $S = [s_1, s_2, \dots, s_M]^T$, where the components of S , satisfy $\sum_{i=1}^M s_i = 1$.

Output:

A permutation $\{x_1, x_2, \dots, x_M\}$ of $\mathcal{A}$.

Method:

begin

$\mathcal{V} := \mathcal{A}$;

    for $i := 1$ to M do

        begin

            Compute the conditional distribution $S_{\mathcal{V}}$ according to (4.1).

            Choose an symbol x_i according to this distribution.

$\mathcal{V} := \mathcal{V} - \{x_i\}$

        endfor

end

End Algorithm S-model

Program 4.1: S-model for permutation generation.

Before we embark on the analysis and recognition phases we shall first introduce the notation that we will be using for the rest of the chapter.

4.2.1 Notation

We use the notation $A_u \rightarrow j$ to denote the fact that the symbol A_u appears at the j^{th} position in the permutation. Furthermore, $A_u, A_v \rightarrow \langle i, j \rangle$ represents the fact that the symbols A_u and A_v appear at the positions i and j in the permutation, respectively. The extensions to this notation, where we are specifying the position of more than two symbols are straightforward and are interpreted in an intuitive fashion.

$A_u \prec A_v$ denotes the event that symbol A_u precedes symbol A_v in a given permutation.

4.3 Analysis of the S-model

We now proceed to state and prove some fundamental properties of the permutations generated by Algorithm S-model.

Theorem 4.1 *Let $\mathcal{S}$ be the (M -dimensional) parameter vector of the S-model. Also let $A_u, A_{v_1}, \dots, A_{v_K}$ be distinct symbols of $\mathcal{A}$. Then the probability that the symbol A_u precedes each one of the symbols of the set $\{A_{v_1}, \dots, A_{v_K}\}$ in any permutation generated by Algorithm S-model, is given by:*

$$P(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}) = \frac{s_u}{s_u + \sum_{j=1}^K s_{v_j}} \quad (4.2)$$

Proof:

We prove this result by induction. Observe that the result stated in Theorem 4.1 must be proved for all (feasible) values of K and M . Since $\{A_{v_1}, \dots, A_{v_K}\}$ can at most include all the symbols of $\mathcal{A} - \{A_u\}$, it follows that $K \leq M - 1$. A traditional induction on K would merely involve proving that the result is true for all K satisfying $K \leq M - 1$. Although M is a constant in our original setting, it must be treated as a parameter, to render the proof valid. The rationale for considering M as a parameter is as follows. For a given M the result could be proved by induction, without thereby implying that the result would continue to be true if the dimensionality of the problem M is changed. Observe that (4.2) states the assertion of the theorem independent of the dimension of the set $\mathcal{A}$.

As a consequence of the above reasoning, it appears that a "two-dimensional induction", i.e. an induction on two variables is unavoidable. However, we shall reduce this into a single parameter induction. This is done as follows. For a given M , we shall prove that (4.2) is valid if it is valid for all values of K and $M = M - 1$. The formal steps of the proof follow.

The theorem is trivially valid if $K = M - 1$. In this case, the denominator of (4.2) essentially consists of the sum $\sum_{i=1}^M s_i$, which is unity. The expression in (4.2) thus simplifies to s_u — which is indeed the probability that the symbol A_u appears before each of the other symbols of $\mathcal{A}$, i.e. in the first position.

We now proceed by induction on the parameter M . Let us assume that the theorem holds for all values of $K \leq M - 1$, for a specific value M_0 of the parameter M . We will establish that the same property holds for $M = M_0 + 1$ as well.

To establish the basis for the induction, we examine the case when $M = 2$. In this case the only meaningful value of K is 1. Since $K = M - 1$, the result indeed holds for the case when $M = 2$. We now proceed with the induction step.

Let $\mathcal{S} = [s_1, s_2, \dots, s_M]^T$, where $M = M_0 + 1$. In order to derive the probability that the symbol A_u precedes each one of the symbols in $\{A_{v_1}, \dots, A_{v_K}\}$, we note that this event happens if:

- (i) A_u appears in the first position.
- (ii) Any symbol not contained in the set $\{A_u\} \cup \{A_{v_1}, \dots, A_{v_K}\}$ appears in the first position and *then* A_u (recursively) precedes $\{A_{v_1}, \dots, A_{v_K}\}$ in the permutation of the remaining $M - 1$ symbols.

Note that the events labelled as (i) and (ii) are mutually exclusive and hence using the notation of Section 4.2.1, from the law of total probability, we have:

$$\begin{aligned}
 & Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}) \\
 &= Pr(A_u \rightarrow 1) + \\
 & \quad \sum_{A_l \notin \{A_u\} \cup \{A_{v_1}, \dots, A_{v_K}\}} Pr(A_l \rightarrow 1) Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}_{\mathcal{A}-\{A_l\}}) \\
 &= s_u + \sum_{A_l \notin \{A_u\} \cup \{A_{v_1}, \dots, A_{v_K}\}} s_l Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}_{\mathcal{A}-\{A_l\}}) \quad (4.3)
 \end{aligned}$$

We note that $\mathcal{S}_{\mathcal{A}-\{A_l\}}$ is a control vector of dimension M_0 , and can be denoted as $[s'_\beta \mid \beta \in \{1, 2, \dots, M_0\} - \{l\}]^T$ and hence each of the probabilities in the RHS of equation (4.3), can be calculated using the induction hypothesis. To illustrate, let us consider the term:

$$Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}_{\mathcal{A}-\{A_l\}}) = \frac{s'_u}{s'_u + \sum_{j=1}^K s'_{v_j}}. \quad (4.4)$$

Notice that $s'_\beta = s_\beta / (1 - s_l)$, since $1 - s_l$ is the normalizing factor for all the probabilities in the control vector $\mathcal{S}_{\mathcal{A}-\{A_l\}}$. Therefore:

$$Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}_{\mathcal{A}-\{A_l\}}) = \frac{s_u}{s_u + \sum_{j=1}^K s_{v_j}}. \quad (4.5)$$

Notice that this probability does not depend on the index l . Combining (4.5) with (4.3), we have:

$$\begin{aligned} Pr(A_u \prec \{A_{v_1}, \dots, A_{v_K}\} \mid \mathcal{S}) &= s_u + \sum_{A_l \notin \{A_u\} \cup \{A_{v_1}, \dots, A_{v_K}\}} s_l \frac{s_u}{s_u + \sum_{j=1}^K s_{v_j}} \\ &= s_u \left(1 + \frac{\sum_{A_l \notin \{A_u\} \cup \{A_{v_1}, \dots, A_{v_K}\}} s_l}{s_u + \sum_{j=1}^K s_{v_j}} \right) \\ &= \frac{s_u}{s_u + \sum_{j=1}^K s_{v_j}} \cdot \left(\sum_{l=1}^{M_0+1} s_l \right) \\ &= \frac{s_u}{s_u + \sum_{j=1}^K s_{v_j}}. \end{aligned} \quad (4.6)$$

Let $f(K, M)$ represent the fact that (4.2) is true for specific values of K and M . Thus, as a consequence of (4.6) we have,

$$\{\forall K \leq M-1\} [f(K, M)] \Rightarrow \{\forall K \leq M-1\} [f(K, M+1)]$$

Since $f(M, M+1)$ is trivially true, we have effectively proved that

$$\{\forall K \leq M-1\} [f(K, M)] \Rightarrow \{\forall K \leq M\} [f(K, M+1)]$$

This completes the proof. □

The above result naturally leads us to the following Corollary.

Corollary 4.1 *In a permutation generated by the S-model, the probability that a symbol A_u precedes another symbol A_v is simply given by $s_u/(s_u + s_v)$.*

Proof:

The result is a special case of (4.2) when $K = 1$. This agrees with the result derived in [75]. □

We shall make extensive use of Corollary 4.1, in deriving the estimates for the control vector $\mathcal{S}$. This aspect is the subject of the next section. However, for the

sake of completeness, in Section 4.6 we present a few additional applications of the S-model. These applications are in areas not directly related to the source recognition problem. In Section 4.6.1 we also study in detail, the application of the S-model to arrive at a presortedness metric for ensembles of data sequences.

We now focus our attention on the estimation of the parameters of the S-model.

4.4 Modeling Distributions Using the S-model

In the following analysis, we shall use the notation $\mathbf{X}$ to denote the vector (of dimension M) representing a permutation of $\mathcal{A}$. Let us further assume that $P(\mathbf{X})$ denotes the actual (unknown) underlying density function and $P_a(\mathbf{X})$ denotes the approximate distribution arrived at, using the "S-Model" of permutation generation. Then the closeness measure between these distributions can be computed using the well known information theoretic metric (given in (1.2)) as:

$$I(P, P_a) = \sum_{\mathbf{X}} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})}$$

In the light of this closeness metric, it is clear that we must choose the approximation $P_a(\mathbf{X})$ in such a way that the closeness measure is minimized. Since the approximating distribution is completely specified by the control vector $\mathcal{S}$, our intention of finding the best approximating density function translates to the problem of finding the best estimate for the vector $\mathcal{S}$. In the following discussion we therefore focus on the aspect of estimating the control vector $\mathcal{S}$ subject to this constraint.

At this point it is useful to write down the expression for the probability of generating a specific permutation $\mathbf{X} = [x_1, x_2, \dots, x_M]$ when the underlying control vector is $\mathcal{S} = [s_1, s_2, \dots, s_M]^T$. To calculate the above probability, we use the *Chain rule*, and the notation of Section 4.2.1, to yield:

$$\begin{aligned} &Pr(x_1, x_2, \dots, x_M) \\ &= Pr(x_1 \rightarrow 1) \cdot Pr(x_2 \rightarrow 2 \mid x_1 \rightarrow 1) \cdot Pr(x_3 \rightarrow 3 \mid x_1, x_2 \rightarrow \langle 1, 2 \rangle) \cdot \\ &\quad \dots Pr(x_M \rightarrow M \mid x_1, x_2, \dots, x_{M-1} \rightarrow \langle 1, 2, \dots, M-1 \rangle). \end{aligned} \quad (4.7)$$

Clearly $Pr(x_1 \rightarrow 1)$ is s_{x_1} . Thus, a specific application of Theorem 4.1 yields the following expressions for the first few factors in (4.7):

$$\begin{aligned} Pr(x_2 \rightarrow 2 \mid x_1 \rightarrow 1) &= \frac{s_{x_2}}{1 - s_{x_1}} \\ Pr(x_3 \rightarrow 3 \mid x_1, x_2 \rightarrow \langle 1, 2 \rangle) &= \frac{s_{x_3}}{1 - s_{x_1} - s_{x_2}} \end{aligned} \quad (4.8)$$

The generalizations of (4.8) for indices larger than three is straightforward and hence combining (4.7) and (4.8) we obtain the expression for the joint probability $Pr(x_1, x_2, \dots, x_M)$ as:

$$Pr(x_1, x_2, \dots, x_M) = \prod_{i=1}^{M-1} \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \quad (4.9)$$

At this juncture, two methods for the estimation of the control vector can be seen. The first method is based on minimizing the closeness metric $I(P, P_a)$. The second method is the well known Maximum Likelihood Estimation (MLE) of parameters [18]. Although superficially these two methods appear to be different, we shall show that they essentially yield the same estimates for the control vector $\mathcal{S}$. We now proceed to state and prove a theorem concerning the "volume" of "sufficient statistics" to be maintained, in order to attempt the calculation of the MLE.

Theorem 4.2 *The maximum likelihood estimate of the control vector $\mathcal{S}$ can be obtained only if we maintain an exponential amount of statistics.*

Proof:

Let $X^1, X^2, \dots, X^N$ denote the N independent samples of permutations of the symbols of $\mathcal{A} = \{A_1, A_2, \dots, A_M\}$, where each symbol of $\mathcal{A}$ is fully identified by its index. We use the notation that α_i^k is the i^{th} symbol in the k^{th} sample permutation X^k . It is useful to recall that the MLE for the control vector $\mathcal{S}$, is that vector which supports the N samples in the strongest manner and thus the likelihood of generating this set of permutations is maximized to yield the best estimate of the parameters.

We can use (4.9) to calculate the probability of generating any one of the sample permutations presented to us. In particular, the probability of generating the sample permutation X^k can be written from (4.9) as:

$$Pr(X^k) = \prod_{i=1}^{M-1} \frac{s_{\alpha_i^k}}{1 - \sum_{j=1}^{i-1} s_{\alpha_j^k}}$$

Therefore, based on the fact that the permutations themselves are independently generated³, the likelihood function $\mathcal{L}$ is given by

$$\mathcal{L} = \prod_{k=1}^N Pr(X^k) = \prod_{k=1}^N \prod_{i=1}^{M-1} \left\{ \frac{s_{\alpha_i^k}}{1 - \sum_{j=1}^{i-1} s_{\alpha_j^k}} \right\} \quad (4.10)$$

In order to simplify the expression for the likelihood function in (4.10), we define the following quantities:

$\mathcal{N}_i^j$	Number of times the integer i occurs in position j
$\mathcal{N}_{ij}$	Number of times the pair (i, j) appears in one of the positions $(1, 2)$ or $(2, 1)$.
$\mathcal{N}_{\alpha_1, \alpha_2, \dots, \alpha_r}$	Number of times the tuple $(\alpha_1, \alpha_2, \dots, \alpha_r)$ or any permutation of it appears in the first r positions of the permutations.

We now proceed to simplify the expression in (4.10). Observe that each of α_i^k is an integer in the range $\{1, 2, \dots, M\}$. For concreteness, let us consider the term s_1 . s_1 occurs in the innermost product of equation (4.10), if the integer 1 appears in the permutation in any one of the positions $\{1 \dots M-1\}$. Similar conclusion holds w.r.t. the other parameters $s_2, \dots, s_M$. Also note that in the denominator of (4.10), the term $(1 - s_i)$ occurs $\mathcal{N}_i^1$ times; $(1 - s_i - s_j)$ occurs $\mathcal{N}_{ij}$ times and so on. Thus, the logarithm of the likelihood function $\mathcal{L}$ defined in (4.10) can be written down as:

$$\begin{aligned} \ell &\triangleq \log \mathcal{L} \\ &= \sum_{j=1}^{M-1} \mathcal{N}_i^j \cdot \log s_i - \sum_{i < j} \log(1 - s_i - s_j) \cdot \mathcal{N}_{ij} \\ &\quad - \sum_{i < j < k} \log(1 - s_i - s_j - s_k) \cdot \mathcal{N}_{ijk} \\ &\quad \dots - \sum \log(1 - s_{i_1} - s_{i_2} \dots - s_{i_{M-2}}) \cdot \mathcal{N}_{i_1, i_2, \dots, i_{M-2}} \end{aligned} \quad (4.11)$$

³Note that this does not imply that the symbols **within** a permutation are independently generated.

We note that $\sum_{j=1}^{M-1} \mathcal{N}_i^j = N - \mathcal{N}_i^M$, and hence, instead of maintaining $O(M^2)$ counters of the form $\mathcal{N}_i^j$, we can maintain exactly M counters which correspond to the frequencies of each of the integers $1, 2, \dots, M$ appearing in the last position in the permutation.

It is easy to see that the number of counters to be maintained for $\mathcal{N}_{\alpha_1, \alpha_2, \dots, \alpha_r}$ is simply given by $\binom{M}{r}$. Hence the total amount of information to be gathered is given by

$$M + \sum_{r=2}^{M-2} \binom{M}{r} = \sum_{r=1}^{M-2} \binom{M}{r} = 2^M - M - 2.$$

This establishes the claim that the number of statistics to maintain is exponential in the parameter M . $\square$

The above result has given an idea of the amount of statistics to be gathered in order to arrive at a MLE for the control vector $\mathcal{S}$. Although this method of arriving at the estimate for the control vector is difficult to implement, it possesses one important characteristic. We will show an important relationship⁴ between the maximum likelihood estimate and the estimate which minimizes the closeness of approximation $I(P, P_a)$.

Theorem 4.3 *Let $\mathcal{E}$ be any event with associated probability $Pr(\mathcal{E})$. If $\hat{Pr}(\mathcal{E})$ is the frequency estimate of the probability $Pr(\mathcal{E})$ obtained using the law of large numbers, then the maximum likelihood estimate obtained as per Theorem 4.2 also minimizes the closeness of approximation $I(P, P_a)$.*

Proof:

Let us now focus our attention to the aspect of minimizing the closeness measure $I(P, P_a)$. The closeness measure $I(P, P_a)$ is:

$$\begin{aligned} I(P, P_a) &= \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} \\ &= \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \log P(\mathbf{X}) - \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \log P_a(\mathbf{X}). \end{aligned} \quad (4.12)$$

---

⁴The result proved below is shown to be true for the case when $\mathbf{X}$ is a permutation. We believe that the result is true universally.

Observe that the first term of (4.12) is independent of the approximation $P_a(\mathbf{X})$. Hence we shall concentrate on the second term of (4.12). Utilizing the expression for the probability of generating a specific permutation under the S-model of permutation generation given in (4.9) the second term of (4.12) can be simplified as follows:

$$\begin{aligned}
 & \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \log P_a(\mathbf{X}) \\
 &= \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \log \left(\prod_{i=1}^{M-1} \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \right) \\
 &= \sum_{\mathbf{X} \in \Pi} P(\mathbf{X}) \cdot \left(\sum_{i=1}^{M-1} \log \frac{s_{x_i}}{1 - \sum_{j=1}^{i-1} s_{x_j}} \right) \\
 &= \sum_{i=1}^M \sum_{j=1}^{M-1} \log s_i \cdot Pr(i \rightarrow j) \\
 &\quad - \sum_{i < j} \log(1 - s_i - s_j) \cdot Pr(i, j \rightarrow \langle 1, 2 \rangle \cup i, j \rightarrow \langle 2, 1 \rangle) \\
 &\quad - \sum_{r=2}^{M-2} \log \left(1 - \sum_{j=1}^r s_{i_j} \right) \cdot \\
 &\quad Pr(\text{some perm. of } i_1, i_2, \dots, i_r \text{ appears in positions } 1, 2, \dots, r)
 \end{aligned}$$

Note that the probabilities of the form $Pr(i \rightarrow j)$ are not known *a priori* and hence they must be estimated from the samples. As per the hypothesis, if we use the frequency estimates obtained by the law of large numbers, the estimates for the corresponding probabilities can be written down as:

$$\begin{aligned}
 Pr(i \rightarrow j) &= \frac{\mathcal{N}_i^j}{N} \\
 Pr(i, j \rightarrow \langle 1, 2 \rangle \text{ or } \langle 2, 1 \rangle) &= \frac{\mathcal{N}_{ij}}{N} \\
 &\vdots
 \end{aligned}$$

With these estimates for the probabilities it is clear that maximizing the term $\sum_{\mathbf{X}} P(\mathbf{X}) \log P_a(\mathbf{X})$ is the same as maximizing the likelihood function derived in (4.11) in Theorem 4.2. Hence the theorem. $\square$

Theorem 4.2 has established that in order to rigorously obtain the MLE for the control vector $\mathcal{S}$, we are required to maintain $O(2^M)$ statistics. From a practical

point of view, this is simply not a viable approach to parameter estimation. We therefore examine whether approximate estimates can be obtained if one is restricted to “collecting” a substantially smaller amount of statistics. Our next result develops the method for estimating the parameters, if only $O(M^2)$ statistics are available.

Theorem 4.4 *Let ${}_uN_v$ represent the number of times the symbol A_u appears before the symbol A_v , in the N (sample) permutations that were observed⁵. If only the frequency counts of the form ${}_uN_v$ are available for all pairs u , and v , then the maximum likelihood estimate for the control vector S can be obtained as the solution of the simultaneous non-linear equations given below:*

$$\sum_{i \neq j} \left\{ \frac{{}_iN_j}{s_i} - \frac{N}{s_i + s_j} \right\} + \lambda = 0 \quad \text{for } 1 \leq i \leq M$$

$$\sum_{i=1}^M s_i = 1. \quad (4.13)$$

Proof:

We emphasize that in this part of the proof, we are not attempting to maximize the likelihood of generating the N sample permutations presented to us. Instead we focus on maximizing the likelihood of explaining the frequencies of the events of the form “symbol A_i precedes symbol A_j ” which are symbolically represented as “ $A_i \prec A_j$ ”.

Consider the event in which the symbol A_i appeared before the symbol A_j for a total of ${}_iN_j$ times. Since the N sample permutations are *statistically independent*, we can write down the probability of occurrence of this event as:

$$Pr(A_i \prec A_j \text{ occurs } {}_iN_j \text{ times})$$

---

⁵ ${}_uN_v$ must not be mistaken for the N_{ij} which was used in the previous theorems. Unlike the former defined here, the latter count refers to the number of times i, j appeared in one of the positions $\langle 1, 2 \rangle$ or $\langle 2, 1 \rangle$.

$$\begin{aligned}
&= \binom{N}{iN_j} [Pr(A_i \prec A_j)]^{N_j} [Pr(A_j \prec A_i)]^{N_i} \\
&= \binom{N}{iN_j} \left(\frac{s_i}{s_i + s_j} \right)^{N_j} \left(\frac{s_j}{s_i + s_j} \right)^{N_i} \\
&\quad \text{By Corollary 4.1} \\
&= \binom{N}{iN_j} \frac{s_i^{iN_j} s_j^{jN_i}}{(s_i + s_j)^N} \tag{4.14}
\end{aligned}$$

The statistics extracted from the data samples consists of counts iN_j , for all $i \neq j$, i.e. $M(M-1)/2$ counters and these are sufficient to evaluate terms of the form (4.14).

In order to arrive at the best estimates for the control vector $\mathcal{S}$, we have to maximize the quantity $Pr[\cap_{i < j} (A_i \prec A_j \text{ occurs } iN_j \text{ times})]$. Assuming that these events are *statistically independent*⁶ we shall consider an approximation to the likelihood function $\mathcal{L}$ (given only the statistics iN_j) as shown below:

$$\begin{aligned}
\mathcal{L} &\triangleq Pr[\cap_{i < j} (A_i \prec A_j \text{ occurs } iN_j \text{ times})] \\
&= \prod_{i < j} Pr[(A_i \prec A_j \text{ occurs } iN_j \text{ times})] \\
&= \prod_{i < j} \binom{N}{iN_j} \frac{s_i^{iN_j} s_j^{jN_i}}{(s_i + s_j)^N} \quad \text{from (4.14)}
\end{aligned}$$

Hence the logarithm of the likelihood function, $\ell \triangleq \log \mathcal{L}$, can be written as:

$$\ell = Z + \sum_{i < j} \{iN_j \log s_i + jN_i \log s_j - N \log(s_i + s_j)\} \tag{4.15}$$

where Z stands for the quantity $\sum_{i < j} \log \binom{N}{iN_j}$.

It is now clear that our aim is to maximize the quantity ℓ subject to the condition that $\sum_{i=1}^M s_i = 1$. This is achieved by using *Lagrange Multipliers* [80] in which if λ is the Lagrange multiplier, the auxiliary function is given by (4.16) below:

$$\sigma = \ell + \lambda \left(\sum_{i=1}^M s_i - 1 \right). \tag{4.16}$$

---

⁶This assumption is strictly not true. A counter example can be easily derived for example when $N = 1$ and $M = 3$.

In order to obtain the maxima of this auxiliary function σ , we merely differentiate it w.r.t. each of the variables $s_1, s_2, \dots, s_M$ and equate the result to 0. Differentiating σ w.r.t. the variable s_i and simplifying yields:

$$\frac{\partial \sigma}{\partial s_i} = \sum_{j \neq i} \left(\frac{i N_j}{s_i} - \frac{N}{s_i + s_j} \right) + \lambda = 0.$$

This set of equations, together with the constraint that $\sum_{i=1}^M s_i = 1$, yield the solution vector $\mathbf{S}$. $\square$

Remark:

It must be noted that the above set of equations are a non-linear set of simultaneous equations involving the s_i 's and the Lagrange multiplier λ . Clearly, an analytic solution is not feasible. We chose to use the generalized form of the Newton-Raphson (NR) method for solving these equations. The essential technique is as follows. Let us assume that the set of simultaneous equations is $f_i(\mathbf{X}) = 0$, for $1 \leq i \leq n$, where $\mathbf{X}$ is an n -dimensional vector. If $\mathbf{X}_0$ is an approximate solution, then the update rule for NR iteration is simply:

$$\mathbf{X}_{k+1} = \mathbf{X}_k - \mathbf{J}^{-1} [f_1(\mathbf{X}_k) \cdots f_n(\mathbf{X}_k)]^T$$

where, $\mathbf{J}$ is the well known Jacobian

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

A first-order estimate for the control vector can be found by examining (4.13). In our simulations, the initial estimate for NR-iteration was the solution obtained by the solving the (linear) simultaneous equations in (4.17).

$$\begin{aligned} \frac{i N_j}{s_1} - \frac{N}{s_1 + s_j} &= 0 \quad j = 2, 3, \dots, M \\ \sum_{i=1}^M s_i &= 1 \end{aligned} \tag{4.17}$$

The reader will recognize that the first $M - 1$ of these equations are obtained by examining the equation of the derivative w.r.t. s_1 and individually equating the bracketed terms to 0. An alternative way to look at these equations is to view that we are merely basing our estimates of the control vector $\mathcal{S}$ on the number of times the symbol A_1 appears before each of the other symbols. It should be also clear that we can use another set of linear equations, in which the coefficients are dependent on the counts of some other symbol A_j preceding the other symbols. Thus we can effectively get M initial approximations which could be different.

4.4.1 Experimental Results

To demonstrate the strength of the procedures used to estimate the control vector, we have performed two types of experiments. These two experiments differ only in the strategy used to generate the sample permutations which are examined by the data analysis procedures.

In the first set of experiments, the model used to generate the permutations is the S-model defined in Section 4.2. Firstly, the number of symbols M in $\mathcal{A}$ is chosen and then a control vector $\mathcal{S}^*$ is chosen randomly. The experiment is subsequently conducted as follows. Sample permutations are generated, based on the control vector $\mathcal{S}^*$. With each sample, the statistic of the form $;N_j$ (defined in Theorem 4.4) are updated. The estimation of the control vector is not attempted after each sample, but rather after a fixed number of samples which was set to 200 in the experiments. The NR method was used to estimate the control vector (denoted as $\hat{\mathcal{S}}$). If this method for solving the system of equations did not converge, we resorted to an explicit numerical optimization procedure to optimize the likelihood function given by (4.15). The method chosen by us to perform the necessary constrained optimization is the David-Fletcher-Powell algorithm, used in conjunction with the interior penalty function [80]. Clearly, in this case, we are interested in observing the convergence of $\hat{\mathcal{S}}$ to $\mathcal{S}^*$ with time.

Figure 4.2 depicts this convergence for the case when M , the number of symbols in each permutation is 6 in which we have used the Euclidean norm between the

vector $\mathcal{S}^*$ and the estimate $\hat{\mathcal{S}}$ as the measure of closeness between these vectors. Figure 4.2 shows the variation of the quantity $|\mathcal{S}^* - \hat{\mathcal{S}}|$ as a function of the number of samples. The results shown in Figure 4.2 are the ensemble average of 25 experiments (consisting of 10,000 samples each) performed with the same control vector $\mathcal{S}^*$. Observe that for the case shown in Figure 4.2, the quantity $|\mathcal{S}^* - \hat{\mathcal{S}}|$ reduced to about 15% of its initial value after about 10,000 samples. In all cases we have studied, we have observed that $\hat{\mathcal{S}}$ converges to the underlying control vector $\mathcal{S}^*$. It must be noted that the rate of convergence of the quantity $|\mathcal{S}^* - \hat{\mathcal{S}}|$ is dependent on the control vector $\mathcal{S}^*$ itself. If some of the components of the control vector are small, some of the counts ${}_iN_j$ are zero until a sufficient number of samples have been generated. This clearly influences the convergence of the estimate $\hat{\mathcal{S}}$ to the underlying vector $\mathcal{S}^*$.

In the second set of experiments, our aim is to judge the effectiveness of the S-model of permutation generation in approximating general distributions. As was done in the case of the earlier set of experiments, we first choose M , the number of symbols in $\mathcal{A}$. To avoid specifying a large number of probability masses, the number M was chosen to be a small number (4 in our case). Having fixed the value of the parameter M , we generated a random distribution of the permutations. In other words, we randomly assign probabilities to each of the $M!$ permutations (subject to the condition that their sum is unity). As opposed to the case in which we were using the S-model of permutation generation, in this case we are not so much interested in the convergence of the estimate $\hat{\mathcal{S}}$ to its final value. In this case the experiment consisted of generating 10,000 sample permutations, based on the chosen distribution and updating the counts ${}_iN_j$ with the arrival of each sample. When the incoming samples were exhausted, the estimation of the control vector was performed. Figure 4.3 (composed of Figures 4.3a and 4.3b) shows the results we have obtained in two specific runs. Both the figures show the underlying distribution (bold shading) and the approximated distribution (mosaic shading). Figure 4.3b shows the more favorable case in which there is a good similarity between the underlying distribution and the estimated distribution. It appears as if whenever adjacent (in

terms of symbolic inversions) permutations have comparable probabilities in the underlying distribution, the S-model yields a good approximation to the underlying distribution.

4.4.2 Numerical Instability

The solution of (4.13) to yield the maximum likelihood estimate for the vector $\mathcal{S}$ obviously entails the solution of M simultaneous non-linear equations. Since these equations have to be solved numerically, this solution automatically inherits the numerical instability of the root solving technique utilized. Thus it is no wonder that whenever the underlying problem is ill-conditioned, the root solving technique does not converge.

It must be noted that unlike traditional problems which involve root solving, the coefficients of the non-linear equations are themselves random variables. If in the course of a particular experiment, a certain event did not occur, the corresponding coefficient would be exactly zero. Thus it is not too astonishing to observe that when the number of samples is small, the matrices involved are singular and the underlying problems are extremely ill-conditioned.

In many of the cases, the NR method leads to a divergent solution. As mentioned earlier, in these cases, rather than solve (4.13), we have opted to solve the underlying optimization problem using the David-Fletcher-Powell algorithm (in conjunction with a penalty function) [80]. We would like to emphasize that our primary contribution is not the root solving or the optimizing algorithm, but the results concerning the existence and properties of the estimates themselves. Clearly, the strategy of evaluating them is merely one of using the well established numerical tools.

4.5 Use of S-model in Source Recognition

We now return to the application of the S-model to the original problem we have on hand, i.e. one of distinguishing between several sources of random permutations. As has been explained earlier, our problem is one of supervised pattern classification. In

Figure 4.2: Convergence of the Estimate of the S-vector.

the training phase of our recognition system, we are presented with a stream Q_{train} , together with information on the source which was used to generate this stream. The “feature extraction” process is now one of estimating the S-parameters for this source, using the estimation procedures explained in Section 4.4. If the number of sources that we need to distinguish is c , the above process can be repeated once for each source. Hence after the “training phase” the recognizer has c estimates $\hat{S}_1, \hat{S}_2, \hat{S}_3, \dots, \hat{S}_c$ for the S-vectors of the c classes.

In the usual pattern recognition system, one is also required to characterize the distribution of features within a given class. In our case this would imply that we ought to determine the distribution of the control vectors $\hat{S}_i | \omega_i$, for $1 \leq i \leq c$. In our setting this is unnecessary since even in the infinite sample case the vector $\hat{S}_i$ is a point in the M -dimensional space and the probability of $\hat{S}_i$ converging to S_i^* can be shown to be unity by a trivial application of the law of large numbers.

For the testing phase, the traditional approach would require that we estimate the control vector $\hat{S}_{test}$ from the test samples and then classify the given test sample based on some discriminant functions. There are two difficulties when we wish to use the same approach in our source recognition problem:

1. If the number of samples in Q_{test} is small, the estimate for $\hat{S}_{test}$ may be not obtainable or grossly inaccurate. We generally encounter the first condition because of singularities in the matrices involved in the estimation process.
2. Any discriminant based method assumes that we can compute the density $P(\hat{S}_{test} | \omega_i)$ for $i = 1, 2, \dots, c$. These probabilities can be computed using the information about class-conditional distributions. As was mentioned above, in our case the estimation of the class-conditional distributions is eliminated by our training procedure.

In our case we have opted not to estimate the control vector $\hat{S}_{test}$ from the test samples but instead to follow the *ab initio* approach: that is, decide the test sample as belonging to the class ω_i if $Pr(\omega_i | Q_{test}) > Pr(\omega_j | Q_{test})$ for all $i \neq j$,

where Q_{test} stands for the sequence of permutations presented to source recognition system. This decision criterion can be easily rewritten as:

$$\text{decide } \omega_i \text{ if } Pr(\omega_i)Pr(Q_{test} | \omega_i) > Pr(\omega_j)Pr(Q_{test} | \omega_j) \text{ for all } i \neq j$$

If we assume that all the classes are equally likely, the above inequality further simplifies to:

$$\text{decide } \omega_i \text{ if } Pr(Q_{test} | \omega_i) > Pr(Q_{test} | \omega_j) \text{ for all } i \neq j \quad (4.18)$$

In effect, this means that the Bayesian classifier can be obtained by essentially utilizing the maximum likelihood rule (or equivalently its logarithmic form). We note that since the individual permutations in the test samples are statistically independent, we can easily compute the likelihood function $Pr(Q_{test} | \omega_i)$ as:

$$Pr(Q_{test} | \omega_i) = \prod_{q \in Q_{test}} Pr(q | \hat{S}_i) \quad (4.19)$$

We now present the experimental results performed in connection with the source recognition problem.

4.5.1 Experimental results

In the first set of experiments which we have performed, the number of elements in a permutation was set to 4, and thus at any instant the recognizer obtains one of twenty-four different permutations. To render the problem non-trivial, we consider a 4-class problem in which we have four independent sources of permutations, all of which are based on the S-model of permutation generation. The control vectors used by the sources (labelled as Classes ω_1 through ω_4) were:

$$\mathcal{S}_1 = [0.4, 0.3, 0.2, 0.1]^T$$

$$\mathcal{S}_2 = [0.1, 0.4, 0.3, 0.2]^T$$

$$\mathcal{S}_3 = [0.2, 0.1, 0.4, 0.3]^T$$

$$\mathcal{S}_4 = [0.3, 0.2, 0.1, 0.4]^T$$

The distributions of these permutations are shown in Figure 4.4.

During the training phase the source recognition system was presented with 1,000 permutations from each class. This information was used to estimate the control vector for each of the four classes. The recognizer was subsequently tested on 200 test cases where each test case consisted of Z sample permutations. The Bayesian classification rule involves computing the likelihood functions given in (4.19) based on the (estimated) control vectors for each of the classes. The class which maximized the likelihood function was chosen by the classifier as the source of the permutations.

After the training phase in which 1,000 samples from each class were used to get the estimates $\{\hat{S}_i\}$, the number of samples, Z , used for each testing operation was varied. When Z was as large as 500 the estimates $\hat{S}_{test}$ that we would have obtained would be reasonably close to the estimate $\hat{S}_i$ of the underlying class. Thus obtaining a high classification accuracy using (4.18) is not too astonishing. Indeed the accuracy that we obtained was a 100% even though we did not estimate $\hat{S}_{test}$ but merely invoked the maximum likelihood classifier described by (4.18). However when the number of samples in a testing operation is decreased, it is clear that the estimate $\hat{S}_{test}$ is increasingly inaccurate and thus the accuracy of such a classifier would diminish. This phenomenon is indeed what we observed but the accuracy is very high even for very small number of test samples. When only 12 samples per test case were used, the accuracy was 95%. Observe that this meant that in this scenario at least half of the 24 permutations were not even seen **once** by the classifier. This accuracy increased to 98% when 18 test samples were used per testing operation. The accuracy was 100% when the number of test samples was greater than 24. The power of the strategy is obvious.

We have also performed analogous experiments when the number of sources was 4, but the number of elements in each permutation (i.e. M) was 6 and 8. For the case when each permutation consisted of 6 elements, the control vectors utilized for the sources (labelled as Classes ω_1 through ω_4) were:

$$S_1 = [0.2424, 0.2121, 0.1818, 0.1515, 0.1212, 0.0909]^T$$

$$S_2 = [0.0322, 0.2580, 0.2258, 0.1935, 0.1612, 0.1290]^T$$

$$\mathcal{S}_3 = [0.0689, 0.0348, 0.2758, 0.2413, 0.2068, 0.1724]^T$$

$$\mathcal{S}_4 = [0.1111, 0.0740, 0.0370, 0.2962, 0.2592, 0.2222]^T$$

Once again, the recognition accuracy was excellent. We noticed that in the case when 10 samples per case were utilized (i.e. only 10 out of 720 permutations were seen by the classifier), the recognition accuracy was an astonishing 99%. The accuracy was 100% when more than 10 samples per test case were presented to the recognizer. The recognition accuracy for the case of 8 elements per permutation was even better. Even when 10 samples per test case were used, we obtained 100% accuracy.

4.6 Applications of the S-model

It can be seen that the S-model can be used generate random permutations, with a wide variety of distributions. This, for example, can be used to derive fast approximate solutions to the classical Traveling Salesman Problem (TSP). This problem belongs to the well-known class of NP-hard problems. However, a very practical approach for solving TSP, based on the idea of “simulated annealing” exists. “Simulated annealing” is essentially a probabilistic global minimum (or maximum) finding procedure. In the case of the TSP, we randomly choose an initial permutation of cities. The procedure to solve this problem maintains information about the shortest route encountered so far. If a different, shorter route is found, the algorithm revises its idea of the best route. The key idea is one of experimenting with several possibilities and eventually converging to the correct minimum (or maximum). Crucial to the simulated annealing family of algorithms is the ability to perturb the current configuration of the system — which in our case would be equivalent to the permutation of cities. These perturbations could be affected by suitably invoking a random permutation generator.

A more specialized application involves the pattern recognition of strings [27]. Consider the problem of a string X , being transmitted over a noisy channel. If the noisy channel induces substitution, insertion and deletion errors, the received

Figure 4.3: Comparison of the Distribution generated by the S-model to the “real” one.

Figure 4.4: Distributions used by the 4 sources.

string, say Y can be used in conjunction with the characteristics of the channel to estimate X_s . Typically, this involves the editing of Y to every X in the dictionary, and techniques are available to achieve fast editing. If, on the other hand, the channel also induces permutation errors (i.e. the channel randomly swaps symbols of X_s), the problem is far more complex. This is due the fact that the editing of Y to every X with substitutions, insertions, deletions and transpositions is an NP-complete problem [50]. In this case, we anticipate that the techniques we present, can be used to perform the pattern recognition. Indeed, the strategy we propose is as follows. We first model the noisy channel as a cascading of two sub-mechanisms, the first of which performs a random permutation and the second which garbles the string using substitution, insertion and deletions. Clearly if the parameters of these model can be extracted using the techniques suggested here, the receiver can try to estimate the string X_s by mimicking the channel to obtain an estimate that yields the minimum probability of error. This is analogous to the method presented in [38] for substitution, insertion, and deletion errors. This proposed solution requires further research.

For a third example, consider a typical operating (or scheduling) system in which M users, $\{A_j \mid 1 \leq j \leq M\}$ submit their jobs according to random priorities $\{z_j(i) \mid 1 \leq j \leq M\}$. Let us assume that the tasks are executed in the decreasing order of their priorities. Thus a typical scheduling sequence consists of a permutation of the set $\{A_j \mid 1 \leq j \leq M\}$. Notice that if the on the i^{th} day the set of priorities are $\{z_j(i) \mid 1 \leq j \leq M\}$, the scheduling sequence for that day will be $\pi(i) \in \Pi$, and one such permutation results from each job submission. Typical questions which the operating system is faced with would involve analyzing the data so as to project the mean position of user A_j in any scheduling sequence. Such questions can answered by modeling the random permutation generator by using the S-model. Observe that this is no hypothetical situation, but is commonplace in every scheduling application.

We now turn our attention to a more significant problem, namely that of assigning a presortedness metric to an ensemble of data sequences.

4.6.1 Ensemble Presortedness Metric

Sorting is a problem of paramount importance in the field of computer science. For concreteness let us assume that the data sequences to be sorted consists of a permutation of the elements in the set $\{A_1, A_2, \dots, A_M\}$, where $A_i < A_j$ whenever $i < j$. The optimal sequential time complexity of sorting a sequence of M elements, is $O(M \log M)$. While many optimal techniques are known, there also exist algorithms (e.g. Quicksort) which are not optimal in the worst case sense. These algorithms can be presented with specific sequence of data elements, for which a running time of $O(M^2)$ results. This phenomenon has led researchers to seek alternate sorting methods which never encounter this worst case quadratic behavior. The key idea motivating this research is to find methods whose time complexity increases “smoothly” as a function of some measure of “presortedness” of the input sequence [14, 61, 62].

The class of “adaptive” sorting methods achieve exactly this goal. These methods monitor some measure of the input sequence, such as the number of inversions [45] or the number of oscillations [51]. Such a measure in essence defines the “distance” of the input data sequence, from its sorted form. Algorithms based on these measures have a time complexity which is $O(M)$ when the “distance” measure is small and approaches the lower bound for sorting as the measure increases [58].

While such adaptive algorithms perform rather well on specific input sequences, an objective comparison of these algorithms with other existing methods (or even among themselves) cannot be achieved by just studying the “distance metrics” of individual sequences. An example will help clarify this point. Let us suppose that the set of data elements to be sorted is $\{1, 2, 3, 4, 5\}$. If an Insertion Sort (IS) is presented with the sequence $\pi_1 = \langle 1, 2, 3, 5, 4 \rangle$ the sorting is achieved by performing exactly one swap and $O(M)$ comparisons. However if the input sequence was $\pi_2 = \langle 5, 4, 3, 2, 1 \rangle$, $O(M^2)$ swaps would be necessary.

As opposed to the IS, let us suppose that the same data sequences were presented to an adaptive scheme, such as the Adaptive HeapSort (AHS) described in [51]. The AHS would take $O(M)$ operations to sort each of the sequences π_1 and π_2 . The

question now before us is to objectively compare IS and AHS.

If the data to be sorted was only of the type π_1 , clearly the IS is superior. IS only makes $O(M)$ comparisons and also avoids the overhead encountered in maintaining and updating the heap used by the AHS. However, if the data set was always of the type π_2 , clearly the AHS is superior. In order to objectively compare the two algorithms, not only should the presortedness metric of a sequence be considered, but the comparison must also involve the probability distribution on the set of permutations on the data set.

Thus our stand in this section is that whereas distance metrics such as Inversions [45], Runs, and especially Oscillations [51] are extremely powerful in quantifying the presortedness of a **single** data sequence, they are incapable of characterizing the presortedness of **ensembles** of sequences. Instead of assigning metrics to each data sequence contained in an ensemble, we assign a single Ensemble Presortedness measure to the ensemble as a whole. This single parameter reflects the divergence of the various sequences in this ensemble from the purely sorted from.

Strictly speaking, given a set of M elements, the probability associated with each possible data sequence can be fully defined by enumerating $M!$ probabilities. Apart from this characterization being unrealistic and impractical, it does not provide a single parameter with which we can compare two ensembles of data sequences. In this light, as alluded to above, we shall present a new metric called the Ensemble Presortedness metric. This metric is completely specified by a single parameter ρ . Given the parameter ρ , the distribution on all possible data sequences is completely described by means of an M – dimensional probability vector. The i^{th} component of this vector is recursively defined in terms of the $(i - 1)^{st}$, and is obtained by merely multiplying the latter by ρ . Indeed if $\rho = 0$, it represents an ensemble consisting only of the sorted sequence. Furthermore we shall show that as ρ increases, the number of inversions increases.

At this point it is appropriate to compare our metric with other metrics that could be envisioned. A simple example of such an ensemble presortedness metric is the average number of inversions in the data sequences found in the ensemble.

The main characteristic of these measures of presortedness is that they quantify the extent of presortedness in an ensemble, but unfortunately they are not capable of *utilizing* the measure generated. Specifically, these metrics cannot be easily integrated into methods for mimicking the statistical characteristics of the ensemble in question, by generating random permutations of data. As opposed to these metrics, our measure forms the basis for a random permutation generation model. As a natural consequence, the measure utilized can be used to generate data sequences which are statistically similar to the ensemble examined. We believe that this concept is new both in the areas of data generation and data analysis.

The approach taken to modeling ensembles of data sequences is to model them as being generated using the S-model of permutation generation. Based on the above discussion, we define s_i as follows:

$$s_i = k\rho^{i-1}$$

Observe that this implies that the complete vector $\mathcal{S}$ is given by:

$$\mathcal{S} = k[1, \rho, \rho^2, \dots, \rho^{M-1}]^T. \quad (4.20)$$

where k is a normalizing factor chosen so as to render $\mathcal{S}$ to be a probability vector. It will be seen that if $\rho < 1$, the components of $\mathcal{S}$ in (4.20) are in the descending order of magnitude. This fact can be used to show that the data sequence $\langle A_1, A_2, \dots, A_M \rangle$ occurs with the highest probability. Thus, as claimed earlier, in the limit as $\rho \rightarrow 0$, the data sequence $\langle A_1, A_2, \dots, A_M \rangle$ appears with probability 1. In other words, the ensemble generated by the model consists only of the sorted list.

On the other hand, if $\rho > 1$, the components of $\mathcal{S}$ are in ascending order. As the value of ρ becomes increasing larger, the last component of $\mathcal{S}$ becomes much larger than the rest, thereby implying that the data sequence $\langle A_M, A_{M-1}, \dots, A_1 \rangle$ will occur with the highest probability. In the limit as $\rho \rightarrow \infty$, this probability reaches the limiting value of unity and in this case the only sequence presented to the sorting algorithm is the reverse of the sorted list.

Since intermediate value of ρ are of interest, we observe that the value $\rho = 1$ corresponds to the case in which the components of $\mathcal{S}$ are all equal. In this case all

the $M!$ data sequences are generated by the ρ -model with equal probability. This corresponds to the uniformity assumption traditionally made in expected case analysis. We now present a few theoretical properties of the data sequences generated according to the single parameter restriction of the S-model.

Lemma 4.1 *Let $\mathcal{S} = [1, \rho, \rho^2, \dots, \rho^{M-1}]$ be the (M -dimensional) control vector of the S-model. Then, in the ensemble of data sequences generated, the expected number of inversions is given by:*

$$E[\text{Inversions}|\rho] = \sum_{j>i} \frac{\rho^{j-i}}{1 + \rho^{j-i}}.$$

Proof:

Let Y_{ij} be an indicator random variable defined which takes on the value 1(0) if element A_i appears before (after) element A_j . Then the number of inversions in a data sequence X is given by:

$$\text{Inversions} = \sum_{j>i} Y_{ji}$$

Hence the expected number of inversions in the data sequences characterized by ρ is:

$$E[\text{Inversions}|\rho] = \sum_{X \in \Pi} \sum_{j>i} Y_{ji} Pr(X). \quad (4.21)$$

We observe that $\sum_X Y_{ji} Pr(X)$ is very simply the probability, $b(j, i)$ that element A_j precedes A_i in a random data sequence. With this understanding, (4.21) can be rewritten as:

$$E[\text{Inversions}|\rho] = \sum_{j>i} b(j, i). \quad (4.22)$$

Using Corollary 4.1, $b(j, i)$ can be written down as:

$$\begin{aligned} b(j, i) &= \frac{\rho^{j-1}}{\rho^{j-1} + \rho^{i-1}} \\ &= \frac{\rho^{j-i}}{\rho^{j-i} + 1}. \end{aligned} \quad (4.23)$$

Substituting (4.23) in (4.22), we obtain the desired result. $\square$

Observe from (4.23) that $b(j, i)$ (for all $j > i$) increases as ρ increases. As a natural consequence, the expected number of inversions also increases. Moreover when $\rho = 0$, $b(i, j) = 0$ whenever $j > i$. Also for $j > i$, $\lim_{\rho \rightarrow \infty} b(j, i) = 1$. This implies that $\lim_{\rho \rightarrow \infty} E[\text{Inversions}|\rho] = \sum_{j>i} 1 = M * (M - 1)/2$. This is obviously the maximum number of possible inversions in a data sequence of length M . The problem of investigating the relationship between the parameter ρ and the expected values of other “presortedness” metrics such as Runs and Oscillations is open for further study.

We now turn our attention to the problem of estimating the parameters from the observed data sequences.

4.6.1.1 Evaluation of ρ , the Ensemble Presortedness Metric

Till now we have focussed on the problem of obtaining data sequences which are generated based on an M -dimensional vector $\mathcal{S}$. We demonstrated how the components of this vector $\mathcal{S}$ can be completely specified in terms of a single parameter ρ , called the Ensemble Presortedness metric. We derived several properties of the data sequences generated under this model. In this section, we concentrate on the aspect of estimating the best value of ρ which describes a given data sequence ensemble. We state, with a mere sketch of proof, our first result in this regard.

Theorem 4.5 *The maximum likelihood estimate of the parameter ρ can be obtained only if we maintain $O(2^M)$ statistics.*

Proof:

This theorem can be proved along the same lines as Theorem 4.2. $\square$

Since it is impractical to maintain an exponential number of counters, we attempt to estimate the parameter ρ by maintaining only $O(M^2)$ statistics for events of the form “element A_u precedes element A_v ”.

Theorem 4.6 *Let ${}_uN_v$ represent the number of times the element A_u appears before the element A_v , in the ensemble of N data sequences that were observed. If only*

the frequency counts of the form ${}_iN_j$ are available for all pairs u , and v , then the maximum likelihood estimate for the parameter ρ can be obtained as the solution of the non-linear equation given below:

$$\sum_{i < j} \frac{j - i}{1 + \rho^{j-i}} = \sum_{i < j} \frac{{}_iN_j}{N} (j - i). \quad (4.24)$$

Proof:

As in the case of Theorem 4.4 in which we attempted to estimate the S -vector, we focus on maximizing the likelihood of explaining the frequencies of the events of the form "element A_i precedes element A_j " (symbolically represented as " $A_i \prec A_j$ ") given only the $\binom{M}{2}$ statistics. For ease of derivation, we let $\mathcal{S} = k[1, \rho, \rho^2, \dots, \rho^{M-1}]^T = [s_1, s_2, \dots, s_M]^T$.

In order to arrive at the best estimates for the control vector $\mathcal{S}$, we have to maximize the quantity $Pr[\cap_{i < j} (A_i \prec A_j \text{ occurs } {}_iN_j \text{ times})]$. Using assumptions analogous to those of Theorem 4.4, and following the proof of Theorem 4.4 almost synchronously, we can write the modified likelihood function $\mathcal{L}$ (given only the statistics ${}_iN_j$) as shown below:

$$\begin{aligned} \mathcal{L} &\triangleq Pr[\cap_{i < j} (A_i \prec A_j \text{ occurs } {}_iN_j \text{ times})] \\ &= \prod_{i < j} Pr[(A_i \prec A_j \text{ occurs } {}_iN_j \text{ times})] \\ &= \prod_{i < j} \binom{N}{{}_iN_j} \frac{s_i^{{}_iN_j} s_j^{{}_jN_i}}{(s_i + s_j)^N}. \end{aligned}$$

Hence the logarithm of the likelihood function, $\ell \triangleq \log \mathcal{L}$, can be written as:

$$\begin{aligned} \ell &= Z + \sum_{i < j} \{ {}_iN_j \log s_i + {}_jN_i \log s_j - N \log(s_i + s_j) \} \\ &\quad \text{(substituting } s_i = k\rho^{i-1}) \\ &= Z + \sum_{i < j} \{ {}_iN_j(i-1) \log k\rho + {}_jN_i(j-1) \log k\rho \\ &\quad - N \log(k\rho^{i-1} + k\rho^{j-1}) \} \end{aligned} \quad (4.25)$$

where Z stands for the quantity $\sum_{i < j} \log \binom{N}{{}_iN_j}$.

It is clear that the MLE for ρ can simply be found by differentiating (4.25) w.r.t. the parameter ρ and setting the result to 0. Hence we obtain,

$$\sum_{i < j} \left\{ \frac{i N_j}{\rho} (i - 1) + \frac{j N_i}{\rho} (j - 1) \right. \\ \left. \frac{N}{(k \rho^{i-1} + k \rho^{j-1})} [(i - 1) k \rho^{i-2} + (j - 1) k \rho^{j-2}] \right\} = 0$$

or,

$$\sum_{i < j} \frac{1}{\rho} (i N_j i + j N_i j - N) - \frac{N}{\rho + \rho^{j-i+1}} \{(i - 1) + (j - 1) \rho^{j-i}\} = 0$$

which simplifies to,

$$\sum_{i < j} \left\{ \frac{i N_j}{N} i + \frac{j N_i}{N} j - 1 \right\} - \frac{1}{1 + \rho^{j-i}} \{(i - 1) + (j - 1) \rho^{j-i}\} = 0$$

or equivalently,

$$\sum_{i < j} \frac{j - i}{1 + \rho^{j-i}} = \sum_{i < j} \frac{i N_j}{N} (j - i).$$

and the theorem is proved. $\square$

Remark: It must be noted that the above equation is a non-linear equation involving ρ . Clearly, an analytic solution is not feasible. In general it can be solved by using a root-solving technique such as the NR method.

4.6.1.2 Experimental Results

Having completely described the model for data sequence generation and given an estimation procedure to evaluate the Ensemble Presortedness metric (ρ), is not out of place for us to examine how effectively this quantity reflects the presortedness of various ensembles. To do this, various ensembles of sequences of data were generated. Each ensemble was characterized by the mean number of inversions. The effect of the mean number of inversions on the MLE of the parameter ρ was studied. The experiments were conducted as follows.

For a given number of elements M we allow at most $MAXINV = M * (M - 1) / 4$ inversions in any random data sequence examined. This number represents half of the maximum number of inversions possible in a list of size M . The ensemble characteristics were fixed by imposing a distribution $Q = [q_0, q_1, q_2, \dots, q_{MAXINV}]^T$ on the number of inversions contained in any particular data sequence. In other

words, q_j represents the probability of a random data sequence possessing exactly j inversions. For our experiments, we chose Q to obey:

$$\begin{aligned} Q &= [q_0, q_1, q_2, \dots, q_{MAXINV}]^T \\ &= A[1, (1 - \delta), (1 - \delta^2), \dots, (1 - \delta)^{MAXINV-1}]^T \end{aligned}$$

where δ is a parameter used to generalize the experiments, and A is a normalizing factor chosen so as to ensure that the components of Q sum to unity ⁷.

The method of random data sequence generation for each ensemble was as follows. For a specific value of M , various ensembles of data sequences were generated, based on values of δ in the interval $[-0.9, 0.9]$. Negative values of δ imply a preference for higher number of inversions, whereas the strictly positive value of δ yield a small number of inversions. $\delta = 0$ is a special case and it represents the case when the number of inversions is uniformly distributed in the interval $[0, MAXINV]$. A random number of inversions R was chosen, based on the distribution Q . This number, R , was then randomly partitioned into an array of integers $[b_i | i = 1, 2, \dots, M]$ such that $\sum_{i=1}^M b_i = R$. Following the procedure outlined in [45, pg.11-12], this array b was used as the *inversion table* to generate the unique data sequence that it corresponds to. Using each such sample data sequence, statistics of the form $\langle N_j \rangle$ were updated. This procedure was repeatedly invoked to generate an ensemble of 10,000 samples. After the ensemble was obtained, the MLE of the parameter ρ was obtained by solving (4.24).

Table 4.3 shows the results for the case in which $M = 8$. Figure 4.5 and Figure 4.6 show similar results for the cases in which M equals 16 and 24 respectively.

Observe that the value of ρ increases monotonically with the increase in the expected number of inversions. When the list is almost sorted, the values of ρ are close to 0. On the other hand, as the expected number of inversions rises to values close to $M * (M - 1)/4$, we find that the estimate of ρ approaches unity. This is just as is expected. The use of ρ as a presortedness metric for the ensemble is obvious.

---

⁷The reader must take care not to confuse between Q and S . Q is the distribution on the number of inversions. S is the distribution for the data generation model.

Figure 4.5: Figure shows the increase in the MLE of ρ as the Expected number of inversions increases. In this case $M = 16$. The maximum number of inversions allowed is 60.

Figure 4.6: Figure shows the increase in the MLE of ρ as the Expected number of inversions increases. In this case $M = 24$. The maximum number of inversions allowed is 138.

δ	Expected number of Inversions	MLE of parameter ρ
0.9	0.1111	1.592e-2
0.8	0.25	3.610e-2
0.6	0.66666	1.002e-1
0.5	0.9995	1.563e-1
0.3	2.2617	3.445e-1
0.2	3.4529	4.751e-1
0.0	7.0	6.918e-1
-0.1	8.72107	7.708e-1
-0.2	10.041	8.214e-1
-0.5	12.034	9.029e-1
-0.7	12.5767	9.229e-1
-0.9	12.889	9.3619e-1

Table 4.3: This table shows the increase in the MLE of the parameter ρ as the expected number of inversions in the ensemble of data sequences increases. In this case, the number of elements in each data sequence is $M = 8$. The maximum number of inversions allowed is 14.

4.7 Summary

In this chapter we have studied the problem of distinguishing between sources that randomly generate strings composed of the symbols of a finite alphabet. For the case when the strings generated are arbitrary, we have presented a simple model called the U-model. An application of this recognition strategy in distinguishing between several keys used in encryption is also presented.

Subsequently, we have studied the problem of distinguishing between sources that generate random permutations. We have defined a model for these sources, making use of a vector $\mathcal{S}$ of the same dimension as the number of elements (i.e., M) in the permutations. We have proved that the MLE for the control vector $\mathcal{S}$ can be obtained if we are prepared to maintain an exponential number of statistics. By maintaining $O(M^2)$ statistics, a technique for obtaining a modified MLE has been derived. Finally, experimental results demonstrating the effectiveness of the S-model in the proposed source recognition strategy have been presented. The recognition

accuracy is excellent, even if the number of permutations presented to the classifier is small[93].

Additionally, we have applied the S-model for permutations to capture the degree of presortedness in an ensemble of data sequences.

Chapter 5

Doubly-Linked Lists

5.1 Introduction

In the preceding chapters, we have considered the application of learning strategies to problems related to pattern recognition. In this chapter and the subsequent ones, we shall study adaptive learning algorithms applicable to data storage and data compression. Specifically, the topic we shall study in this chapter concerns heuristics for Doubly-Linked Lists (DLLs). We begin with the study of ergodic heuristics. A cursory view of ergodic heuristics was presented in Section 1.2.1 of Chapter 1. In the interest of brevity, we shall not repeat the basics here. We now specify the nature of queries being made against the DLL, and outline the goal we wish to attain.

A DLI, D , consists of the set of N records $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ stored in some order. Accesses to the record R_i are made from the left and right with probabilities s_{iL} and s_{iR} respectively. These probabilities obey the constraint that: $\sum_{i=1}^N (s_{iL} + s_{iR}) = 1$.

We assume that the record accesses are statistically independent and identically distributed. We also assume that the starting position of the search for a record is independent of the record itself. In our case, we will always commence the search

for an element from the end at which the element was requested.

Our aim is to maintain D in an approximately optimal order with respect to the expected cost. Before we investigate any heuristics to reorganize the list, it is convenient to understand what constitutes an optimal arrangement.

5.1.1 The optimal order in a DLL

Let π be a given arrangement of the records. The position of the record R_i in π will be denoted by $\pi^{-1}(i)$. We note that the cost of locating the record R_i is $\pi^{-1}(i)$ if the search commences from the left and is $N - \pi^{-1}(i) + 1$ for searches from the right. Hence the conditional expected cost can be written down as:

$$\begin{aligned} E[\text{Cost} | \text{Configuration is } \pi] &= \sum_{i=1}^N s_{iL} \pi^{-1}(i) + \sum_{i=1}^N s_{iR} (N - \pi^{-1}(i) + 1) \\ &= \sum_{i=1}^N (s_{iL} - s_{iR}) \pi^{-1}(i) + \sum_{i=1}^N s_{iR} (N + 1) \end{aligned} \quad (5.1)$$

Notice that in (5.1), the second term is independent of the permutation π , and hence the optimal order is the permutation which arranges the list in the decreasing order of the difference $s_{iL} - s_{iR}$. Since we assume that the access distribution $\mathcal{S}$ is unknown *a priori*, we will consider the use of heuristics which endeavor to minimize the expected cost.

5.1.2 Scope of this Chapter

In this chapter we study both ergodic and absorbing DLL heuristics. In connection with ergodic heuristics, we establish a sufficient condition for one DLL heuristic to be better than another in the sense of lower expected cost. We make use of this condition to compare the DLL heuristics SWAP and Move-to-End (MTE) which are obtained by extending the well known Singly-Linked lists (SLL) heuristics Transpose (TR) and Move-to-Front (MTF) respectively which were briefly introduced in Section 1.2.1. Specifically, we identify sufficient conditions under which SWAP has

a lower expected asymptotic cost than the MTE. The necessary conditions are still being investigated.

We subsequently turn our attention to absorbing rules for reorganizing DLLs. We present an absorbing scheme which is based on using one additional counter, and is a generalization of the SLL absorbing heuristic proposed in [76]. For a restricted family of distributions, this heuristic is shown to be asymptotically optimal. Thus, for these distributions, the probability of converging to the optimal order can be made as close to unity as desired.

It is rather astonishing to discover that the families of distributions for which the above absorbing heuristic is asymptotically optimal happens to be a superset of the family of distributions under which SWAP outperforms the MTE heuristic. We begin our discussion by considering ergodic heuristics.

5.2 Heuristics for DLL

We initiate the study of DLL heuristics by introducing a few definitions.

Definition 5.1 *A memoryless heuristic for a DLL is described by a set of permutations given by $\tau = \{\tau_i^L, \tau_i^R\}$. The permutation τ_i^L (τ_i^R) is applied whenever the element located at position i in the list is accessed from the left (right) [60].*

Definition 5.2 *Let $C_\tau(S)$ denote the expected cost resulting from the use of the heuristic τ when the access distribution is $S = [s_{1L}, s_{1R}, s_{2L}, s_{2R}, \dots, s_{NL}, s_{NR}]^T$. A heuristic τ is said to be more efficient than another heuristic σ if $C_\tau(S) < C_\sigma(S)$ for all distributions S .*

The definition for memoryless heuristics for DLLs suggests a rather straightforward way in which SLL heuristics can be applied to DLLs. Let $\rho = \{\rho_i\}$ be a SLL heuristic. A very simple approach to mapping ρ to become a DLL heuristic is to set $\tau_i^L = \rho_i = \tau_i^R$. In other words, the data reorganization is insensitive to the direction of access. This mapping is termed as an **Undirected mapping** [69]. In [69] Ng and Oommen have derived the sufficient conditions to be satisfied by distributions

for which the ranking of heuristics in the SLL domain is preserved when they are mapped to the DLL domain by an undirected mapping.

The other approach to synthesizing a DLL heuristic is by using a mapping called a **directed mapping**. In this approach, whenever a record at position i is accessed from the left, we apply the permutation $\tau_i^L = \rho_i$. On the other hand if the access is from the right, we merely apply a “reflection” of the permutation ρ_i . In other words, we apply the same permutation with the assumption that the list starts at the right end and terminates at the left.

A simple application of this technique transforms the well known MTF rule for SLLs to the MTE rule described in [60]. For fixing ideas, let us denote the order of elements in the DLL D , by the permutation $\langle R_{i_1}, R_{i_2}, \dots, R_{i_N} \rangle$ where $\langle i_1, i_2, \dots, i_N \rangle$ is a permutation of the integers $1 \dots N$. The MTE operates in the following fashion: if the element R_{i_j} is accessed from the left ($j \neq 1$), the accessed element is moved to the left end of the list, resulting in the configuration $\langle R_{i_j}, R_{i_1}, R_{i_2}, \dots, R_{i_{j-1}}, R_{i_{j+1}}, \dots, R_{i_N} \rangle$. On the other hand, if the element R_{i_j} is accessed from the right (and $j \neq N$, i.e. the element is not already at the end), the accessed element is moved to the right end, yielding the list: $\langle R_{i_1}, R_{i_2}, \dots, R_{i_{j-1}}, R_{i_{j+1}}, \dots, R_{i_N}, R_{i_j} \rangle$.

In this chapter, we study the effect of applying a directed mapping to the Transposition heuristic to generate the SWAP heuristic for DLLs. The SWAP heuristic was proposed by Ng and Oommen [69]. For the sake of clarity, a description of the heuristic is given below.

As before, let the order of elements in the DLL be $\langle R_{i_1}, R_{i_2}, \dots, R_{i_N} \rangle$. In the SWAP heuristic, whenever a record R_{i_j} is accessed from the left, the list is reorganized by swapping R_{i_j} with $R_{i_{j-1}}$. We note that no swap is performed if $j = 1$ and the element being accessed is at the left end of the list D . On the other hand, if the search for the element R_{i_j} commenced at the right end, we swap R_{i_j} with $R_{i_{j+1}}$, unless $j = N$. The special case identified corresponds to the case when the R_{i_j} happens to be the last element in the list.

Rivest showed that the TR heuristic always has a lower expected cost when compared to the MTF heuristic[81]. Intuitively there is good reason to believe that SWAP should perform better than the MTE rule. While we conjecture that this is indeed true, a rigorous proof is by no means simple. The focus of this chapter is therefore, to establish sufficient conditions which will ensure that SWAP is better than MTE.

In the following derivations, we will assume, without loss of generality that $s_{iL} - s_{iR} \geq s_{jL} - s_{jR} \forall i < j$. In other words, the optimal order of the records is $\langle R_1, R_2, \dots, R_N \rangle$. We shall make extensive use of the fact that the probability of a given record R_i preceding another record R_j under a heuristic τ (denoted by $b_\tau(i, j)$) determines the cost of using τ . We now state and prove a sufficient condition which allows us to compare the efficiency of any two arbitrary heuristics.

Theorem 5.1 *Let τ and σ be two DLL heuristics with the property that $b_\tau(i, j) > b_\sigma(i, j) \forall i < j$. Then τ is more efficient than σ .*

Proof:

We denote the expected cost produced by the heuristics τ and σ as $C_\tau(\mathcal{S})$ and $C_\sigma(\mathcal{S})$ respectively. We can write down the expected cost under the heuristic τ as:

$$\begin{aligned} C_\tau(\mathcal{S}) &= \sum_{i=1}^N s_{iL} \left\{ 1 + \sum_{j \neq i} b_\tau(j, i) \right\} + \sum_{i=1}^N s_{iR} \left\{ 1 + \sum_{j \neq i} b_\tau(i, j) \right\} \\ &= \sum_{i=1}^N s_{iL} \left\{ 1 + \sum_{j \neq i} b_\tau(j, i) \right\} + \sum_{i=1}^N s_{iR} \left\{ 1 + \sum_{j \neq i} (1 - b_\tau(j, i)) \right\} \\ &= \sum_{i=1}^N \{s_{iL} + N s_{iR}\} + \sum_{i=1}^N \sum_{j \neq i} s_{iL} b_\tau(j, i) - \sum_{i=1}^N \sum_{j \neq i} s_{iR} b_\tau(j, i). \end{aligned}$$

Since the first term is independent of i , we denote it using the constant K , and thus:

$$\begin{aligned} C_\tau(\mathcal{S}) &= K + \sum_{i < j} \{s_{iL} b_\tau(j, i) + s_{jL} b_\tau(i, j) - s_{iR} b_\tau(j, i) - s_{jR} b_\tau(i, j)\} \\ &= K + \sum_{i < j} \{(s_{jL} - s_{jR}) - (s_{iL} - s_{iR})\} b_\tau(i, j) + (s_{iL} - s_{iR}). \end{aligned}$$

The expression for $C_\sigma(\mathcal{S})$ has an identical form, and hence we can write down the difference between $C_\tau(\mathcal{S})$ and $C_\sigma(\mathcal{S})$ as:

$$C_{\tau}(\mathcal{S}) - C_{\sigma}(\mathcal{S}) = \sum_{i < j} \{(s_{jL} - s_{jR}) - (s_{iL} - s_{iR})\} (b_{\tau}(i, j) - b_{\sigma}(i, j)). \quad (5.2)$$

Since we have assumed that $b_{\tau}(i, j) > b_{\sigma}(i, j)$ and that $s_{iL} - s_{iR} > s_{jL} - s_{jR} \forall i < j$, the RHS of (5.2) is negative and the result is proved. $\square$

5.2.1 Comparison of the SWAP and MTE heuristics

Because of Theorem 5.1, a sufficient condition for SWAP to perform better than MTE is that the probability of record R_i being before another record R_j (as viewed from the left) should be greater for the SWAP heuristic than for the MTE heuristic, whenever $i < j$. We wish to identify distributions for which $b_{\text{SWAP}}(i, j) > b_{\text{MTE}}(i, j) \forall i < j$.

Ng and Oommen analyzed the Markov chain for the SWAP heuristic, making use of the fact that it is *time reversible* [69, 82]. The relation obeyed by the stationary distribution of the chain representing the SWAP heuristic is stated in Lemma 5.1 below [69]:

Lemma 5.1 *Consider the two list configurations $\langle R_{i_1}, R_{i_2}, \dots, R_i, R_{i+1}, \dots, R_{i_N} \rangle$ and $\langle R_{i_1}, R_{i_2}, \dots, R_{i+1}, R_i, \dots, R_{i_N} \rangle$. The ratio of the asymptotic probabilities of these two configurations is given by:*

$$\frac{\text{Pr}(\langle R_{i_1}, R_{i_2}, \dots, R_i, R_{i+1}, \dots, R_{i_N} \rangle)}{\text{Pr}(\langle R_{i_1}, R_{i_2}, \dots, R_{i+1}, R_i, \dots, R_{i_N} \rangle)} = \frac{s_{iL} + s_{i+1R}}{s_{iR} + s_{i+1L}}. \quad (5.3)$$

Proof:

The proof is due to Ng and Oommen and can be found in [69]. $\square$

We now formally define the families of distributions examined in this chapter.

Definition 5.3 *An access distribution $\mathcal{S} = [s_{1L}, s_{1R}, s_{2L}, s_{2R}, \dots, s_{NL}, s_{NR}]^T$ satisfying*

$$s_{iL} - s_{iR} > s_{jL} - s_{jR} \forall i < j$$

is said to be **Exclusive**¹ if the following condition holds whenever $i < j$:

$$s_{kL}(s_{iL} - s_{jL}) + s_{kR}(s_{jR} - s_{iR}) + (s_{iL}s_{jR} - s_{jL}s_{iR}) > 0. \quad \forall k \neq i, j. \quad (5.4)$$

Theorem 5.2 Let S be an exclusive distribution, and R_i and R_j be any two records of $\mathcal{R}$, where $i < j$. Also let $\langle k_1, k_2, \dots, k_m \rangle$ denote some permutation of m of the integers in the set $\{1, 2, \dots, N\} - \{i, j\}$. Then, the following result holds:

$$\frac{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_m} R_j L_2 \rangle)}{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_m} R_i L_2 \rangle)} \geq \frac{s_{iL} + s_{jR}}{s_{jL} + s_{iR}}$$

where L_1 and L_2 are arbitrary prefix and suffix lists respectively, made up of the elements of $\mathcal{R} - \{R_i, R_j\} - \{R_{k_1}, R_{k_2}, \dots, R_{k_m}\}$.

Proof:

For the sake of clarity, we denote:

$$\xi_{ij}(\langle k_1, k_2, \dots, k_m \rangle) = \frac{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_m} R_j L_2 \rangle)}{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_m} R_i L_2 \rangle)}$$

The theorem trivially holds when $m = 0$ (by (5.3)) and thus the basis step of an inductive proof is established. We will prove the general result by induction, assuming in the inductive step that the theorem is true for $m = l - 1$. That is, we assume that:

$$\xi_{ij}(\langle k_1, k_2, \dots, k_{l-1} \rangle) \geq \frac{s_{iL} + s_{jR}}{s_{jL} + s_{iR}}$$

Now, the ratio of the stationary probabilities of the two states in which records R_i and R_j are separated by l elements can be written down using the chain rule as:

$$\xi_{ij}(\langle k_1, k_2, \dots, k_l \rangle)$$

¹The name Exclusive is used merely for want of a better characterizing property of the family of distributions. Indeed if the condition (5.4) is satisfied, it turns out that ratios of probabilities in which records R_i and R_j are separated by any number of elements has, as its lower bound, the analogous ratio in which records R_i and R_j are adjacent. Thus, in a sense, the influence of the intervening records can be ignored.

$$\begin{aligned}
&= \frac{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)}{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)} \\
&\quad \cdot \frac{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)}{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)} \\
&\quad \cdot \frac{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_l} R_j L_2 \rangle)}{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_l} R_i L_2 \rangle)} \\
&= \xi_{ij}(\langle k_1, k_2, \dots, k_{l-1} \rangle) \cdot \frac{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_{k_l} R_j L_2 \rangle)}{Pr(\langle L_1 R_i R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)} \\
&\quad \cdot \frac{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)}{Pr(\langle L_1 R_j R_{k_1} R_{k_2} \dots R_{k_{l-1}} R_{k_l} R_i L_2 \rangle)} \\
&= \xi_{ij}(\langle k_1, k_2, \dots, k_{l-1} \rangle) \cdot \left[\frac{s_{jR} + s_{k_l L}}{s_{jL} + s_{k_l R}} \cdot \frac{s_{iL} + s_{k_l R}}{s_{iR} + s_{k_l L}} \right] \quad (\text{by (5.3)}) \quad (5.5)
\end{aligned}$$

Let us denote the term within the square brackets of (5.5) as f . The difference between the numerator and the denominator of f can be easily seen to be:

$$s_{k_l L}(s_{iL} - s_{jL}) + s_{k_l R}(s_{jR} - s_{iR}) + (s_{iL}s_{jR} - s_{jL}s_{iR})$$

Because of the assumption that $\mathcal{S}$ is an *exclusive* distribution, the quantity f is strictly greater than 1. Hence, (5.5) can be rewritten as follows:

$$\xi_{ij}(\langle k_1, k_2, \dots, k_l \rangle) \geq \xi_{ij}(\langle k_1, k_2, \dots, k_{l-1} \rangle). \quad (5.6)$$

The theorem follows because of the inductive hypothesis. $\square$

Theorem 5.3 For all exclusive distributions $\mathcal{S}$, $b_{SWAP}(i, j) \geq b_{MTE}(i, j) \forall i < j$, and hence, for these distributions the SWAP heuristic is more efficient than the MTE.

Proof:

Our proof will be a generalization of the proof used by Rivest to prove that TR is better than MTF [81, 82]. We shall show that for all Exclusive distributions, $b_{SWAP}(i, j) \geq b_{MTE}(i, j)$, for $1 \leq i < j \leq N$. We then have:

$$\begin{aligned}
&\frac{b_{SWAP}(i, j)}{b_{SWAP}(j, i)} \\
&= \frac{\sum_{l < m} \sum_{k' \neq i} Pr(\langle R_{k_1}, \dots, R_{k_l}, R_i, R_{k_{l+1}}, \dots, R_{k_m}, R_j, R_{k_{m+1}}, \dots, R_{k_{N-2}} \rangle)}{\sum_{l < m} \sum_{k' \neq j} Pr(\langle R_{k_1}, \dots, R_{k_l}, R_j, R_{k_{l+1}}, \dots, R_{k_m}, R_i, R_{k_{m+1}}, \dots, R_{k_{N-2}} \rangle)} \quad (5.7)
\end{aligned}$$

Notice that in (5.7), the summations are taken over all permutations $k_1, k_2, \dots, k_{n-2}$ of the integers $1, 2, \dots, i-1, i+1, \dots, j-1, j+1, \dots, N$, for all $1 \leq l < m \leq N$. Thus from (5.7) a weak lower bound for the LHS of (5.7) can be immediately written as:

$$\frac{b_{SWAP}(i, j)}{b_{SWAP}(j, i)} \geq \min_{1 \leq l < m \leq N} \left[\frac{\Pr(\langle R_{k_1}, \dots, R_{k_l}, R_i, R_{k_{l+1}}, \dots, R_{k_m}, R_j, R_{k_{m+1}}, \dots, R_{k_{N-2}} \rangle)}{\Pr(\langle R_{k_1}, \dots, R_{k_l}, R_j, R_{k_{l+1}}, \dots, R_{k_m}, R_i, R_{k_{m+1}}, \dots, R_{k_{N-2}} \rangle)} \right]. \quad (5.8)$$

Because of Theorem 5.2, we immediately conclude that:

$$\frac{b_{SWAP}(i, j)}{b_{SWAP}(j, i)} \geq \frac{s_{iL} + s_{jR}}{s_{jL} + s_{iR}}, \quad (5.9)$$

which implies that:

$$b_{SWAP}(i, j) \geq \frac{s_{iL} + s_{jR}}{s_{iL} + s_{jR} + s_{jL} + s_{iR}}. \quad (5.10)$$

Observe that the expression on the RHS of (5.10) is exactly $b_{MTE}(i, j)$, as derived in [60]. Hence we have proved that:

$$b_{SWAP}(i, j) \geq b_{MTE}(i, j) \quad \forall i < j.$$

This information, together with Theorem 5.1, establishes the superiority of the SWAP heuristic, when compared to MTE, for all Exclusive distributions. $\square$

It only remains to show that entire families of Exclusive distributions exist. Two such families are listed below.

Corollary 5.1 *Consider the families of distributions in which $s_{iL} > s_{jL}$ and $s_{jR} > s_{iR}$ for all $i < j$. A distribution of this type is said to be **Dually Monotonic**. All Dually Monotonic distributions are Exclusive and hence, the SWAP heuristic is more efficient than the MTE heuristic for such query distributions.*

Proof:

By virtue of the conditions satisfied by a dually monotonic distribution, we can easily deduce that $s_{iL} - s_{jL} \geq 0$ and $s_{jR} - s_{iR} \geq 0$ for all $i < j$. Moreover, whenever $i < j$, we trivially observe that $s_{iL}s_{jR} \geq s_{iR}s_{jL}$. In other words, this distribution satisfies the condition defined in (5.4), and hence all dually monotonic distributions are also Exclusive. By Theorem 5.3, it follows that for this family of access distributions, SWAP performs better than MTE. $\square$

As an example of a dual monotonic distribution, consider the following distribution in which the access probabilities for the left and right directions are based on Zipf's law. Specifically, the probability of accessing a record R_i (for $i = 1, 2, \dots, N$) from the left and right are given by:

$$\begin{aligned} s_{iL} &= \frac{c_1}{i} \\ s_{iR} &= \frac{c_2}{N - i + 1} \end{aligned}$$

Notice that the constants c_1 and c_2 must be chosen in such a manner that $\sum_{i=1}^N (s_{iL} + s_{iR}) = 1$.

Corollary 5.2 *Consider the family of distributions in which the total probability of accessing record R_i ($1 \leq i \leq N$) is given by p_i . These probabilities obey the relation $p_i > p_j$ for all $i < j$. Furthermore, the conditional probability of accessing a record R_i from the left is p (where $p > 0.5$) independent of the index i . We call this family of distributions **Directionally Biased**. Directionally biased distributions are Exclusive, and hence, the SWAP is more efficient than MTE.*

Proof:

We note that as a result of conditions specified in the statement of the corollary, the probabilities s_{iL} and s_{iR} can be expressed as $p_i p$ and $p_i(1 - p)$ respectively, where p_i represents the total probability of accessing record R_i . For this family of distributions, the condition for their being Exclusive, remains to be verified. Indeed, the LHS of condition (5.4) can be written down as:

$$p_k(p_i - p_j)p^2 + p_k(p_j - p_i)(1 - p)^2 + p_i p_j \{p(1 - p) - p(1 - p)\}$$

$$\begin{aligned}
&= p_k(p_i - p_j) \{p^2 - (1 - p)^2\} \\
&= p_k(p_i - p_j)(2p - 1).
\end{aligned}$$

It is clear that because of the condition that $p > 0.5$, this family qualifies as an Exclusive distribution. The claim immediately follows. $\square$

Note that we have proved the superiority of SWAP relative to MTE, only for Exclusive distributions. Since the TR heuristic is known to be better than the MTF for all access distributions, we expect SWAP to be always better than MTE. This result is by no means trivial to prove. Whenever $b_{SWAP}(i, j) > b_{MTE}(i, j)$ can be established for all $i < j$, SWAP has been shown to be the superior heuristic. However, it is an easy task to construct examples of distributions in which $b_{SWAP}(i, j) < b_{MTE}(i, j)$ at least for one pair of indices i, j where $i < j$. Hence to compare SWAP and MTE, we apparently have to deal with the explicit expression for cost itself and not the expression for the asymptotic pairwise precedence probabilities. This problem is currently open and stated as a conjecture.

Conjecture 5.1 *For all access distributions, the SWAP heuristic has a lower expected cost than the MTE heuristic.*

5.3 Absorbing schemes for DLLs

In the preceding section, we have examined the memoryless heuristics SWAP and MTE for DLLs. These schemes are representative of the class of ergodic schemes because the Markov chain associated with the list configurations is ergodic (i.e. irreducible and aperiodic). In the equilibrium state, the probability of the list being in one its $N!$ configurations can be computed as the stationary probability vector of the transition matrix describing the scheme. Thus, even in the limit, the list can be in one of the $N!$ permutations. These schemes are clearly to be preferred if the environment is non-stationary. Thus, even if the distribution of the queries changes with time, the data reorganization scheme is capable of following these changes.

If the environment is known to be stationary, it is better to use an absorbing scheme, which gradually transforms the list until an absorbing state is reached. After this, no further transformations in the list take place. If the probability of converging to the optimal order can be made exactly unity, all subsequent queries will benefit from the lowest possible cost. If this probability cannot be made exactly unity, we would like to use a heuristic which allows us to attain a value that is “sufficiently high”. This would ensure that the expected cost is close to the optimal one.

We shall now present an absorbing strategy which moves each data element exactly once. In other words, once an element is moved, it is never moved again. After each of the elements in the list has been moved once, the list is considered to have converged to its “final” configuration. In the following section, we explain this procedure in detail and derive some of its asymptotic properties.

5.3.1 Absorbing-MTE

We have seen earlier that the optimal configuration of records is one which arranges the records in the descending order of the difference $s_{iL} - s_{iR}$. Maintaining estimates of the difference in access probabilities clearly requires additional space proportional to the number of elements in the list. Instead of using $O(N)$ such counters, our scheme is based on the use of *one* additional counter, and a parameter M which determines the accuracy of the scheme.

Let $\mathcal{V}$ be the set of elements that have not been moved from their initial positions. $\mathcal{V}$ is initially set to $\mathcal{R}$ (the set of all records in the list) and is progressively reduced until it is empty. At this point, the data reorganization algorithm is said to have converged, and further reorganizations will not take place with subsequent accesses.

Whenever an element R_i is accessed and $R_i \notin \mathcal{V}$ (i.e. the record R_i has already been moved from its initial position) although the contents of R_i are presented to the user, the data reorganization strategy completely ignores this access. We now consider the case when $R_i \in \mathcal{V}$.

To aid in the understanding of the scheme, let us imagine that a new “conceptual” list is being formed as we decide on the final positions of the individual records of $\mathcal{R}$. For simplicity of exposition we shall consider that the new list being formed consists of two sublists, *Front* and *Tail* to be ultimately concatenated.

Initially *Front* and *Tail* are both empty. If a record R_i is accessed M consecutive times from the *left*, it is appended to the end of the list *Front*. As alluded to above, we note that these M consecutive accesses to R_i could be interleaved by any number of accesses to the elements of $\mathcal{R}$ that have been already moved. Analogously, if the record R_i is accessed M consecutive times from the *right*, we add R_i to the list *Tail* as *its* first element. The intuitive explanation is that elements being accessed very frequently from the left are moved to belong to the sublist *Front* and hence appear at the beginning of the reorganized list. On the other hand, elements being accessed more frequently from the right are placed at the head of the list *Tail* and consequently appear near the end of the overall concatenated list. When all the elements of the original list have been moved once, the scheme has converged and we can replace the original list by the new list consisting of $\textit{Front} \cup \textit{Tail}$, where $\cup$ is taken to represent the concatenation operator. Figure 5.1 illustrates the operation of this algorithm, for an example list consisting of 5 elements. The details of the sequence in which the elements are moved and the direction in which they are accessed consecutively is also shown in Figure 5.1. Observe that the initial ordering of the records is $\langle R_3, R_5, R_1, R_2, R_4 \rangle$ and the final ordering is $\langle R_1, R_2, R_3, R_4, R_5 \rangle$.

For concreteness, we present the data reorganization strategy formally as Algorithm Absorbing_DLL_1 in Program 5.1.

Element Accessed M times	Direction	<i>Front</i>	<i>Tail</i>
R_1	Left	$\langle R_1 \rangle$	$\langle \rangle$
R_5	Right	$\langle R_1 \rangle$	$\langle R_5 \rangle$
R_4	Right	$\langle R_1 \rangle$	$\langle R_4, R_5 \rangle$
R_2	Left	$\langle R_1, R_2 \rangle$	$\langle R_4, R_5 \rangle$
R_3	Left	$\langle R_1, R_2, R_3 \rangle$	$\langle R_4, R_5 \rangle$

Figure 5.1: Figure shows how the sublists *Front* and *Tail* are gradually built, as elements of the DLL are moved. The initial order of elements in the DLL is $\langle R_3, R_5, R_1, R_2, R_4 \rangle$. The final ordering is $\langle R_1, R_2, R_3, R_4, R_5 \rangle$.

Algorithm Absorbing_DLL.1

Input:

A sequence of queries presented by the procedure GetQuery and an integer M

Output:

The reorganized DLL

Memory Requirements:

- (i) Two memory locations: z_1, z_2 . z_1 stores the index of the last accessed record. z_2 stores the number of times the record R_{z_1} has been consecutively accessed, if accesses to the elements of $\mathcal{R} - \mathcal{V}$ are ignored.
- (ii) Two memory locations 'Dir' and 'Previous_dir'. 'Dir' takes on values +1(-1) for left(right) respectively

Method:

```

 $\mathcal{V} := \mathcal{R}$ ;
 $z_1 := 0$ ; /* Maximum uncertainty */
 $z_2 := 0$ ;
repeat
  GetQuery ( $R_i$ , Dir);
  if ( $R_i \in \mathcal{V}$ ) then
    if (( $z_1 = i$ ) and (Dir = Previous_dir)) then
       $z_2 := z_2 + \text{Dir}$ ;
    else
      if ( $z_1 \neq 0$ ) then

```

```

         $z_1 := z_2 := 0;$ 
    else
         $z_1 := i;$ 
         $z_2 := \text{Dir};$ 
    endif
endif
if ( $z_2 = M$ ) then
     $\text{Front} := \text{Front} \cup \{R_i\};$ 
     $\mathcal{V} := \mathcal{V} - \{R_i\};$ 
elseif ( $z_2 = -M$ ) then
     $\text{Tail} := \{R_i\} \cup \text{Tail};$ 
     $\mathcal{V} := \mathcal{V} - \{R_i\};$ 
endif
endif
Previous_dir = Dir
until ( $\mathcal{V} = \{\}$ );
NewList := Front  $\cup$  Tail;

```

End Algorithm Absorbing_DLL_1

Program 5.1: Absorbing scheme for DLLs.

The explanation given above is only to aid in the conceptual understanding of the algorithm. To implement the scheme as presented in Algorithm Absorbing_DLL_1 would require additional $O(N)$ storage elements to build the “NewList”. To avoid the overhead of $O(N)$ storage, we would like to update the list “in place”. This modified version of the algorithm will be presented in Section 5.3.2, after we have presented some of its theoretical properties.

Lemma 5.2 *Algorithm Absorbing_DLL_1 is absorbing. Moreover, the total number of list reorganizing operations performed is exactly N .*

Proof:

Obvious. □

We now investigate the conditions under which the above scheme converges to the optimal list order. We shall show that for a family of distributions $\{\mathcal{S}\}$, the

above scheme is asymptotically optimal. To aid in the proof, we introduce some additional notation.

The access probability will be denoted by $\mathcal{S}$. Let $\mathcal{V}$ be any subset of $\mathcal{R}$. We define the conditional access probability vector $\mathcal{S}|\mathcal{V}$ as the vector of normalized probabilities in which only the quantities corresponding to the elements of the set $\mathcal{V}$ have non-zero values. That is:

$$\mathcal{S}|\mathcal{V} = \{s'_{iL}, s'_{iR}\}.$$

where,

$$\begin{aligned} s'_{iL} = s'_{iR} &= 0 \quad \text{if } R_i \notin \mathcal{V} \\ s'_{iL} &= \frac{s_{iL}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} \quad \text{if } R_i \in \mathcal{V} \\ s'_{iR} &= \frac{s_{iR}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} \quad \text{if } R_i \in \mathcal{V} \end{aligned} \quad (5.11)$$

Observe that $\mathcal{S}$ is exactly equivalent to $\mathcal{S}|\mathcal{R}$.

As usual, we shall denote the event “record R_u precedes record R_v ” as $R_u < R_v$.

We make use of a key result in Markov Chains to compute the absorption probabilities. We state this result below.

Lemma 5.3 *Let F be the transition matrix of a homogeneous finite Markov chain where F_{ij} is the probability of making the transition from state i to state j . Then, x_{iA} , the probability of being absorbing into an absorbing state A , starting from a transient state i is given by the following recursive equation:*

$$x_{iA} = F_{iA} + \sum_{j \in \tau} F_{ij} x_{jA}. \quad (5.12)$$

where τ represents the set of all transient states.

Proof:

Can be found in [36]. □

We now state and prove our main result which computes the asymptotic probability of the Algorithm Absorbing_DLL_1 converging to the optimal order.

Theorem 5.4 *Let $\mathcal{S}$ be the access probability vector. Then the total probability that a record R_u precedes another record R_v in the final list obtained by Algorithm Absorbing-DLL-1 is given by:*

$$Pr(R_u \prec R_v) = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}.$$

Proof:

Without loss of generality, we assume that $s_{uL}, s_{uR}, s_{vL}, s_{vR}$, are all non-zero.

For the indices $u, v \in \{1, 2, \dots, N\}$, let $\xi_{uv}(\mathcal{V})$ be the event that either R_u or R_v is the element which is selected from the set $\mathcal{V}$ to be moved, where $\mathcal{V} \subseteq \mathcal{R}$. Also, let $\eta_{uv}(i)$ denote the event that the i^{th} element moved consists either of R_u or R_v . As a result of the data reorganization scheme, we have the simple relation that:

$$Pr[R_u \prec R_v | \eta_{uv}(N - |\mathcal{V}| + 1)] = Pr[R_u \prec R_v | \xi_{uv}(\mathcal{V})]. \quad (5.13)$$

Since $\mathcal{V}$ is the set of elements which have not been moved to their final positions in the list, the conditional access probabilities are defined by (5.11). We now introduce the following probabilities:

$$\begin{aligned} p_L &= \frac{s_{uL}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} & p_R &= \frac{s_{uR}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} \\ q_L &= \frac{s_{vL}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} & q_R &= \frac{s_{vR}}{\sum_{R_j \in \mathcal{V}} (s_{jL} + s_{jR})} \end{aligned}$$

Note that p_L, p_R, q_L, q_R are the conditional probabilities of accessing R_u, R_v from the left and right respectively. We now proceed to compute $Pr[R_u \prec R_v | \xi_{uv}(\mathcal{V})]$. In order to compute this quantity, we first define the Markov chain for the data reorganization strategy under the condition that R_u or R_v is the record to be selected for the move to its final position. The Markov chain is in state $\phi(n)$ at time instant n , where:

$$\phi(n) = \begin{cases} U_i & \text{if } z_1 = u, z_2 = i \\ U_{-i} & \text{if } z_1 = u, z_2 = -i \\ V_i & \text{if } z_1 = v, z_2 = i \\ V_{-i} & \text{if } z_1 = v, z_2 = -i \\ \theta_0 & \text{if } z_1 = 0, z_2 = 0 \\ \theta_i & \text{if otherwise} \end{cases}$$

Figure 5.2: The figure depicts the Absorbing Markov Chain used to model the operation of Algorithm Absorbing_DLL1.

Clearly the Markov chain represents the contents of the memory locations z_1, z_2 . θ_0 represents the state of maximum uncertainty in which there is insufficient information about which of the records R_u or R_v should be moved first. Also θ_t represents the state in which the elements of $\mathcal{V} - \{R_u, R_v\}$ have been accessed at least once (but not M consecutive times).

The transition probabilities of the Markov Chain are given by the stochastic matrix F given below: ($1 \leq i \leq M - 1$):

$$\begin{aligned}
 F_{\theta_0 U_i} = F_{U_i U_{i+1}} &= p_L & F_{U_i \theta_0} &= 1 - p_L \\
 F_{\theta_0 U_{-i}} = F_{U_{-i} U_{-(i+1)}} &= p_R & F_{U_{-i} \theta_0} &= 1 - p_R \\
 F_{\theta_0 V_i} = F_{V_i V_{i+1}} &= q_L & F_{V_i \theta_0} &= 1 - q_L \\
 F_{\theta_0 V_{-i}} = F_{V_{-i} V_{-(i+1)}} &= q_R & F_{V_{-i} \theta_0} &= 1 - q_R \\
 F_{\theta_0 \theta_t} &= 1 - p_L - p_R - q_L - q_R \\
 F_{\theta_t \theta_0} &= 1 \text{ } (\theta_t \text{ is a reflecting barrier}) \\
 F_{U_M U_M} &= 1 & F_{U_{-M} U_{-M}} &= 1 \\
 F_{V_M V_M} &= 1 & F_{V_{-M} V_{-M}} &= 1
 \end{aligned}$$

The Markov Chain is shown pictorially in Figure 5.2. We define the following absorption probabilities, for values of i in the set $\{-M, -(M-1), \dots, 1, 2, \dots, M-1, M\}$.

$$\begin{aligned}
 a_i &= Pr[\phi(\infty) = U_M | \phi(0) = U_i; \xi_{uv}(\mathcal{V})] \\
 b_i &= Pr[\phi(\infty) = V_{-M} | \phi(0) = U_i; \xi_{uv}(\mathcal{V})] \\
 c_i &= Pr[\phi(\infty) = U_M | \phi(0) = V_i; \xi_{uv}(\mathcal{V})] \\
 d_i &= Pr[\phi(\infty) = V_{-M} | \phi(0) = V_i; \xi_{uv}(\mathcal{V})]
 \end{aligned}$$

Notice that a_i denotes the conditional probability of getting absorbed into the state U_M given that we start in the state U_i (and $\xi_{uv}(\mathcal{V})$). The quantities b_i, c_i , and d_i can be given a similar interpretation. Convergence to either U_M or V_{-M} indicates that the record R_u will appear before R_v in the final list. Also note that the following boundary conditions are obeyed by a_i, b_i, c_i, d_i :

$$\begin{aligned}
a_M &= 1 & a_{-M} &= 0 \\
b_M &= 0 & b_{-M} &= 0 \\
c_M &= 0 & c_{-M} &= 0 \\
d_M &= 0 & d_{-M} &= 1
\end{aligned} \tag{5.14}$$

Since the algorithm starts off in the state θ_0 , we are interested in computing the probability of reaching one of the states U_M or V_{-M} . We define the following quantities:

$$\begin{aligned}
\Delta_{0u} &= \Pr[\phi(\infty) = U_M | \phi(0) = \theta_0; \xi_{uv}(\mathcal{V})] \\
\Delta_{0v} &= \Pr[\phi(\infty) = V_{-M} | \phi(0) = \theta_0; \xi_{uv}(\mathcal{V})]
\end{aligned}$$

Clearly, $\Delta_0 = \Delta_{0u} + \Delta_{0v}$ is the quantity we wish to compute. Now, using Lemma 5.3, the recurrence relations obeyed by a_i, b_i, c_i and d_i can be written down as ($1 \leq i \leq M-1; 1 \leq -j \leq M-1$):

$$\begin{aligned}
a_i &= p_L a_{i+1} + (1 - p_L) \Delta_{0u} \\
a_j &= p_R a_{j-1} + (1 - p_R) \Delta_{0u} \\
b_i &= p_L b_{i+1} + (1 - p_L) \Delta_{0v} \\
b_j &= p_R b_{j-1} + (1 - p_R) \Delta_{0v} \\
c_i &= q_L c_{i+1} + (1 - q_L) \Delta_{0u} \\
c_j &= q_R c_{j-1} + (1 - q_R) \Delta_{0u} \\
d_i &= q_L d_{i+1} + (1 - q_L) \Delta_{0v} \\
d_j &= q_R d_{j-1} + (1 - q_R) \Delta_{0v}
\end{aligned} \tag{5.15}$$

Solving (5.15) subject to the boundary conditions given by (5.14), we obtain the following expressions for a_i, b_i, c_i, d_i for $i = \pm 1$:

$$\begin{aligned}
a_1 &= p_L^{M-1} (1 - \Delta_{0u}) + \Delta_{0u} \\
a_{-1} &= \Delta_{0u} (1 - p_R^{M-1}) \\
b_1 &= \Delta_{0v} (1 - p_L^{M-1}) \\
b_{-1} &= \Delta_{0v} (1 - p_R^{M-1})
\end{aligned}$$

$$\begin{aligned}
c_1 &= \Delta_{0u}(1 - q_L^{M-1}) \\
c_{-1} &= \Delta_{0v}(1 - q_R^{M-1}) \\
d_1 &= \Delta_{0v}(1 - q_L^{M-1}) \\
d_{-1} &= q_R^{M-1}(1 - \Delta_{0v}) + \Delta_{0v}
\end{aligned} \tag{5.16}$$

Thus, Δ_0 can be written down as:

$$\begin{aligned}
\Delta_0 &= p_L a_1 + p_R a_{-1} + q_L c_1 + q_R c_{-1} + p_L b_1 + p_R b_{-1} + q_L d_1 + q_R d_{-1} \\
&\quad + (1 - p_L - p_R - q_L - q_R) \Delta_0.
\end{aligned} \tag{5.17}$$

Substituting the expressions in (5.16) into (5.17), and simplifying, we obtain the following result:

$$\Delta_0 = \frac{p_L^M + q_R^M}{p_L^M + p_R^M + q_L^M + q_R^M}. \tag{5.18}$$

Since p_L, p_R, q_L, q_R have the same denominator, we can rewrite (5.18) as:

$$Pr[R_u \prec R_v | \xi_{uv}(\mathcal{V})] = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}. \tag{5.19}$$

Combining (5.19) with (5.13), we have proved that for all $1 \leq i \leq M - 1$,

$$Pr[R_u \prec R_v | \eta_{uv}(i)] = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}. \tag{5.20}$$

Because of the fact that the events $\eta_{uv}(i)$ for $1 \leq i \leq M - 1$ are mutually exclusive and collectively exhaustive, the unconditional (total) probability that the record R_u precedes R_v can be written down using the laws of total probability as:

$$Pr[R_u \prec R_v] = \sum_i Pr[R_u \prec R_v | \eta_{uv}(i)] Pr[\eta_{uv}(i)].$$

Since $Pr[R_u \prec R_v | \eta_{uv}(i)]$ is independent of i (by (5.20)), and $\sum_i Pr[\eta_{uv}(i)] = 1$, we obtain:

$$Pr[R_u \prec R_v] = Pr[R_u \prec R_v | \eta_{uv}(i)] = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}.$$

This completes the proof. $\square$

We now investigate the conditions under which the probability of the above scheme converging to the optimal order can be made arbitrarily close to 1 (by increasing M , the parameter of the scheme). To study this, observe that it is sufficient to attempt to make $Pr(R_i \prec R_j)$ tend to 1, as $M \rightarrow \infty$. Our next result examines the distributions which force this limit to unity.

Theorem 5.5 *Let $\mathcal{S}$ be an access distribution. Then the necessary and sufficient conditions that Algorithm Absorbing_DLL_1 converges to the optimal order with probability 1 as $M \rightarrow \infty$ is:*

$$\max(s_{iL}, s_{jR}) > \max(s_{jL}, s_{iR}) \quad \forall i < j. \quad (5.21)$$

Proof:

We first prove that the conditions stated in (5.21) are sufficient to ensure that Algorithm Absorbing_DLL_1 converges to the optimal order with a probability of unity. By Theorem 5.4, the probability of a record R_i preceding another record R_j is given by:

$$Pr(R_i \prec R_j) = \frac{s_{iL}^M + s_{jR}^M}{s_{iL}^M + s_{iR}^M + s_{jL}^M + s_{jR}^M}.$$

The expression for $Pr(R_i \prec R_j)$ is of the form:

$$Pr(R_i \prec R_j) = \frac{1}{1 + \delta},$$

where,

$$\delta = \frac{s_{jL}^M + s_{iR}^M}{s_{iL}^M + s_{jR}^M}.$$

In the following proof, we assume for convenience that s_{iL} and s_{jR} are distinct, implying that there is a unique maximum. The same assumption will be made with

regard to the access probabilities s_{jL} and s_{iR} . The case when $s_{iL} = s_{jR}$ and/or $s_{jL} = s_{iR}$ can be treated very similarly, without changing the conclusion. Now,

$$\begin{aligned}\delta &= \frac{s_{jL}^M + s_{iR}^M}{s_{iL}^M + s_{jR}^M} \\ &= \frac{\max(s_{jL}^M, s_{iR}^M)}{\max(s_{iL}^M, s_{jR}^M)} \cdot \frac{1 + \frac{\min(s_{jL}^M, s_{iR}^M)}{\max(s_{jL}^M, s_{iR}^M)}}{1 + \frac{\min(s_{iL}^M, s_{jR}^M)}{\max(s_{iL}^M, s_{jR}^M)}}\end{aligned}$$

Clearly,

$$\begin{aligned}\lim_{M \rightarrow \infty} \frac{\min(s_{jL}^M, s_{iR}^M)}{\max(s_{jL}^M, s_{iR}^M)} &= 0 \\ \lim_{M \rightarrow \infty} \frac{\min(s_{iL}^M, s_{jR}^M)}{\max(s_{iL}^M, s_{jR}^M)} &= 0\end{aligned}$$

and hence,

$$\lim_{M \rightarrow \infty} \delta = \lim_{M \rightarrow \infty} \frac{\max(s_{jL}^M, s_{iR}^M)}{\max(s_{iL}^M, s_{jR}^M)} = \lim_{M \rightarrow \infty} \left[\frac{\max(s_{jL}, s_{iR})}{\max(s_{iL}, s_{jR})} \right]^M = 0.$$

Hence δ tends to 0 as $M \rightarrow \infty$, and consequently, $Pr(R_i < R_j)$ tends to 1 as $M \rightarrow \infty$. Note that because of the input conditions, this property holds for all $i < j$.

Let P^* be the probability of converging to the optimal order. Then,

$$\begin{aligned}P^* &= Pr[\cap_{i < j} (R_i < R_j)] \\ &\geq \prod_{i < j} Pr[R_i < R_j].\end{aligned}\tag{5.22}$$

Since we have already established that each of the terms of the RHS of (5.22) tend to 1 as M tends to ∞ , $P^* \rightarrow 1$ as $M \rightarrow \infty$. The fact that (5.21) is necessary, can be proved along similar lines. The proof is omitted for the sake of brevity. $\square$

It is useful to point out that the necessary and sufficient condition specified in (5.21) is not merely of theoretical interest. Indeed, various families of distributions satisfy this condition. In fact, we shall show that the families of distributions presented in Corollary 5.1 and Corollary 5.2 indeed satisfy condition (5.21).

Corollary 5.3 *Let S be a dual monotonic access distribution obeying the conditions of Corollary 5.1. Then, the probability of Algorithm Absorbing_DLL_1 converging to the optimal order approaches 1 as $M \rightarrow \infty$.*

Proof:

Because of the conditions specified in Corollary 5.1 we have the conditions $s_{iL} > s_{jL}$ and $s_{jR} > s_{iR}$, whenever $i < j$. Combining both these inequalities, we obtain $\max(s_{iL}, s_{jR}) > \max(s_{jR}, s_{iR})$, whenever $i < j$. This information, together with Theorem 5.5 establishes our claim. $\square$

We now state our final result.

Corollary 5.4 *Let S be a Directionally Biased Distribution, as defined in Corollary 5.2. Then for this distribution, Absorbing_DLL_1 is asymptotically optimal.*

Proof:

The input conditions are : $p_i > p_j$ for all $i < j$, and that the conditional probability of accessing any record from the left is $p > 0.5$. Since $p > 0.5$, it is clear that $p > p_j/(p_i + p_j)$; as a result we conclude that:

$$\max(s_{iL}, s_{jR}) = \max(p_i p, p_j(1 - p)) = p_i p. \quad (5.23)$$

We note that $\max(s_{jL}, s_{iR})$ can be $p_j \cdot p$ or $p_i(1 - p)$. In the first case, the term in (5.23) is greater, since $p_i > p_j$. In the second case, the difference between $p_i p$ and $p_i(1 - p)$ is $p_i(2p - 1)$ which is greater than 0, because $p > 0.5$. Hence, in either case, the conditions of Theorem 5.5 are met and the required result follows. $\square$

Remark:

1. The conditions imposed by Corollary 5.4 are rather strict. In fact it is easy to show that Absorbing_DLL_1 is optimal for a family of distributions for which the following (weaker) conditions are satisfied:

$$\begin{aligned} s_{iL} &> s_{jL} \quad \forall i < j \\ s_{iL} &> s_{iR} \quad 1 \leq i \leq N. \end{aligned}$$

2. An example distribution which does not satisfy the conditions specified in Theorem 5.5 is given by (5.24) below. For this distribution, the DLL will not converge to the optimal order even if parameter M tends to infinity.

$$\begin{aligned}s_{iL} &= c + \frac{k}{2}i \\ s_{iR} &= \frac{c}{2} + k \cdot i\end{aligned}\tag{5.24}$$

where c, k are positive constants chosen so that $c > kN$. Of course, c and k must be chosen so that the components of S add to unity.

5.3.2 Implementation Considerations

In Section 5.3.1 we had presented Algorithm Absorbing_DLL_1 in a manner that simplified the exposition. The only drawback of implementing the algorithm verbatim from the pseudocode presented is that it necessitates the maintenance of two auxiliary lists *Front* and *Tail* whose total length gradually grows to N , the size of the original list. This was explicitly clarified in Section 5.3.1.

There are two reasons for not implementing the procedure as described. That first is that it requires $O(N)$ storage for maintaining the auxiliary lists. The second, and more important reason is that for the period in which the ultimate positions of only a few records have been decided, subsequent queries do not benefit from the (possibly) improved placement of these records. Thus an “in place” update strategy is desirable, and the following strategy answers both these concerns.

The simple modification to Algorithm Absorbing_DLL_1 is effected by maintaining the sublist *Front* to be the initial part of the list (viewed from the left) and making the sublist *Tail* overlap with the last portion of the DLL (see Figure 5.3). In Figure 5.3, notice that the pointer *EndOfFront* locates the end of the sublist *Front*; it actually points to the element which follows the last element in the sublist *Front*. The pointer *StartOfTail* points to the head of the sublist *Tail*. The elements of the DLL between the positions *EndOfFront* and *StartOfTail* are those elements whose final positions have not been determined.

Figure 5.3: A DLL being reorganized.

Whenever Algorithm Absorbing_DLL.1 decides that an element is to be appended to the sublist *Front*, it is merely removed from its current position and inserted at the position indicated by *EndOfFront*. Analogously, the goal of adding an element as the head of the sublist *Tail* is achieved by merely inserting the element just before the pointer *StartOfTail*. Notice that in our original Algorithm Absorbing_DLL.1, we needed to determine if the record being accessed belongs to the set $\mathcal{V}$ of already moved records. In the modified form of Algorithm Absorbing_DLL.1, it is clear that all the elements of the DLL between its head and *EndOfFront*, as well as the elements encountered between *StartOfTail* and the end of the DLL, belong to the set of already moved records. Thus the set $\mathcal{V}$ is implicitly maintained.

These details are formalized in Algorithm Absorbing_DLL.2. In this algorithm, we assume that the operations of adding an element after *EndOfFront* and adding an element just before *StartOfTail* are elementary operations, and their corresponding pointer manipulations are omitted in the interest of brevity. We have presented an example of the operation of Algorithm Absorbing_DLL.2 in Figure 5.4.

Algorithm Absorbing_DLL.2**Input:**

A sequence of queries presented by the procedure GetQuery and an integer M

Output:

The reorganized DLL

Memory Requirements:

- (i) Two memory locations: z_1, z_2 . z_1 stores the index of the last accessed record. z_2 stores the number of times the record R_{z_1} has been consecutively accessed, if accesses to the elements of $\mathcal{R} - \mathcal{V}$ are ignored.
- (ii) Two memory locations 'Dir' and 'Previous_dir'. 'Dir' takes on values +1(-1) for left(right) respectively
- (iii) Pointers EndOfFront and StartOfTail.

Additional Routines

We assume that the following elementary procedures are available to the Algorithm.

1) InsertBefore (L, CurrElem, Here)

This procedure inserts the element 'CurrElem' in the DLL 'L', before the element pointed to by 'Here'

2) Search (L, Dir, CurrElem, Boundary, FoundWithinBounds)

This procedure searches the DLL 'L' for the element 'CurrElem', using the direction specified by 'Dir'. The Boolean variable FoundWithinBounds is set to 'true' if the required element was found before the pointer 'Boundary' was encountered. It is set to 'false' otherwise. This boolean need not be stored as additional information in each element, but can be trivially computed as explained in Section 5.3.2.

3) DeleteElem(L, CurrElem)

Deletes the element 'Currelem' from the DLL 'L'. If the element being deleted happens to be at the boundary of one of the sublists *Front* or *Tail* then the appropriate pointer, i.e. EndOfFront or StartOfTail will have to be updated as well.

Method:

$\mathcal{V} := \mathcal{R};$

$z_1 := z_2 := 0; /* \text{Maximum uncertainty} */$

EndOfFront := PtrToHead(L);

```

StartOfTail := NIL;
repeat
  GetQuery ( $R_i$ , Dir);
  if (Dir = LEFT) then
    Search (L, LEFT,  $R_i$ , EndOfFront, RiHasMoved)
  else
    Search (L, RIGHT,  $R_i$ , StartOfTail, RiHasMoved);
  if (not RiHasMoved) then
    if (( $z_1 = i$ ) and (Dir = Previous_dir)) then
       $z_2 := z_2 + \text{Dir}$ ;
    else
      if ( $z_1 \neq 0$ ) then
         $z_1 := z_2 := 0$ ;
      else
         $z_1 := i$ ;
         $z_2 := \text{Dir}$ ;
      endif
    endif
  endif
  if ( $z_2 = M$ ) then
    DeleteElem (L,  $R_i$ );
    InsertBefore (L,  $R_i$ , EndOfFront);
  elseif ( $z_2 = -M$ ) then
    DeleteElem (L,  $R_i$ );
    InsertBefore (L,  $R_i$ , StartOfTail);
  endif
endif
Previous_dir := Dir
until (EndOfFront = StartOfTail);

```

End Algorithm Absorbing_DLL_2

Program 5.2: This program shows a practical implementation of the Algorithm Absorbing_DLL_1 presented in Program 5.1.

(a) The initial List configuration

(b) Record R1 moved to the Front

(c) Record R5 moved to the Tail

(d) Record R4 moved to the Tail

(e) Record R2 moved to the Front

Figure 5.4: This example illustrates the operation of Algorithm Absorbing-DLL2 in a simple case involving 5 records. That is, the set of records $\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5\}$. It is assumed that this order corresponds to the optimal order of placing this records. The initial order of the records is $\langle R_3, R_5, R_1, R_2, R_4 \rangle$. The sequence of records moved is exactly the same as that shown in Figure 5.1.

5.4 Summary

In this chapter we have studied the problem of dynamically reorganizing the elements of a DLL. We have compared two DLL heuristics, namely the MTE and the SWAP. These heuristics (MTE and SWAP) can be derived as generalizations from the well known SLL heuristics MTF and TR, by the application of directed mappings.

Since the TR heuristic is superior to the MTF heuristic for SLLs, it is natural to believe that SWAP should be more efficient than MTE for all access distributions. A general proof of this conjecture remains open. In this chapter we have identified a sufficient condition for one DLL heuristic to yield a lower expected cost than another. This criterion merely examines the pairwise record precedence probabilities for the two heuristics under consideration. Since SWAP and MTE are the heuristics being compared, we have identified families of distributions for which this sufficient condition is satisfied, and consequently for all these distributions, SWAP is shown to be asymptotically more efficient than MTE.

We have also proposed an absorbing scheme for reorganizing the elements of a DLL. The proposed scheme utilizes a single parameter, denoted as M . In this scheme, each data element is moved into its final position in the reorganized list if it accessed M consecutive times. Again, it has been shown that for two rather general families of distributions, the probability of our scheme converging to the optimal order can be made as close to unity as desired.

We note in passing that the family of distributions for which algorithm Absorbing-DLL-1 is shown to be asymptotically optimal, happens to be a superset of the family of distributions for which SWAP has been shown to be more efficient than MTE.

Chapter 6

Singly-Linked Lists and Non-Stationary Queries

6.1 Overview of Problem

In the previous chapter, we studied Doubly-Linked Lists (DLLs) under the conventional model of query generation. Observe that in our study of DLLs, the requests were generated independent of the preceding queries, based on the same (time-invariant) distribution. In such a case, the query system is said to be stationary. In this chapter, we wish to study the effects of relaxing these assumptions for the case of Singly-Linked Lists (SLLs). In particular, we study two models of non-stationary query generation. In the first model, we model the query generator *itself* as a Markov Chain. In the second model, we permit the query distribution at each time instant to be a random vector. The well-known Move-to-Front (MTF) heuristic is analyzed under both these models of query generation.

Before we pursue the case of non-stationary queries, we begin with an overview of the analysis strategies employed to study the behavior of SLL heuristics, when the query distribution is time-invariant and the heuristic being utilized is ergodic. In this case, the distribution is represented by the vector $\mathcal{S} = [s_1, s_2, \dots, s_N]^T$, where s_i represents the probability of accessing the record R_i . As already mentioned in Section 1.2.1, the optimal order which minimizes the expected retrieval cost corresponds to one which arranges the records in the descending order of their access

probabilities.

Several heuristics for SLLs were briefly introduced in Section 1.2.1 of the introductory chapter. In the interest of brevity, we shall not repeat them here, but instead, directly proceed to the study of ergodic heuristics. When the SLL reorganizing heuristic is ergodic, as mentioned in the introductory chapter, even asymptotically, the SLL can be in any one of $N!$ states. Thus the behavior of any of the ergodic heuristics is completely characterized by an ergodic Markov Chain and the analysis typically consists of tackling the chain consisting of $N!$ states. Let us denote the set of permutations by $\Pi = \{\pi_i \mid i = 1, 2, \dots, N!\}$ and denote the steady state probability that the Markov Chain is in the state π_i by $Pr(\pi_i)$. If $Cost(\pi_i)$ represents the expected retrieval cost, given that the elements are in the permutation π_i , and C_τ the asymptotic expected cost of retrieval as per a reorganization strategy τ , then C_τ can be written down as:

$$C_\tau = \sum_{i=1}^{N!} Pr(\pi_i) \cdot Cost(\pi_i). \quad (6.1)$$

Since the stationary probabilities for the MTF and the Transposition (TR) rules are analytically obtainable [30, 32, 81], the closed form expressions for the asymptotic expected cost can easily be obtained.

It must be noted that the above analysis schemes rely on the *convergence* of the underlying *Homogeneous* Markov Chain. The convergence properties of the underlying chains can be inferred only if the transition probabilities are independent of time. When the transition probabilities vary with time, predicting the convergence is an extremely difficult problem and the literature available is scanty.

Figure 6.1 shows the Markov Chain for the case when the linear list being accessed has 3 elements and the MTF heuristic is used to reorganize the list. If S , the vector representing the access probabilities is time varying, the transition probabilities of the Markov Chains are also time varying, and hence even the question of the *existence* of a stationary distribution is unsolved. Furthermore, even if the stationary distribution for such a chain can be shown to exist, the technique for evaluating it is unknown except for some extremely trivial or straightforward cases.

Figure 6.1: The Markov Chain for the list with 3 elements. The MTF heuristic is used to reorganize the list with every access.

The only reported list organizing strategy in which the underlying Markov chain is non-homogeneous is the one due to Oommen and Hansen [72]. Their scheme is a variation of the well known MTF rule. In the scheme reported in [72], the accessed element R_i is moved to the front of the list with a probability $f(n) = a^n$ (where $a < 1$) at the time instant n . In other words, $f(n)$ is progressively decreased so that the chain is finally absorbing. The underlying non-homogeneous Markov Chain has the property that the transition matrix can be decomposed into the sum of two matrices, from each of which the term a^n can be factored out. This, augmented with the fact that both these matrices have the same diagonalizing matrix, leads to the solution of the asymptotic distribution. Although these computations are both messy and involved, the techniques used in [72] are novel primarily because it represents one of the few cases in which non-homogeneous chains can be solved explicitly. The important aspect to note is that the underlying chain associated with the *algorithm* is non-homogeneous, even though the user's query distribution is time invariant. In other words, the non-homogeneity is not an inherent property of the underlying *problem*. The non-homogeneity has been introduced *merely because of the heuristic* which is stochastic, and alters its behavior with time.

This chapter of the thesis endeavors to tackle the problem in which the user's query distribution itself is unknown and yet non-stationary. That is, in this chapter we shall consider the effect of having the access probabilities **themselves** change with time. The implications of this are two-fold. First of all, in this case, the chain is non-homogeneous because of the inherent property of the problem solved and not a consequence of the heuristic used. The second implication is that although the non-stationarity may be a consequence of the problem on hand, by a suitable transformation in the solution space, this solution can be obtained by the analysis of an underlying *homogeneous* Markov chain. Both these concepts will be clarified presently.

We first define the concept of "non-stationarity" within the context of this problem. In other words, we need to restrict the types of variations in distributions, as time elapses. We shall present two models for query generators whose query distri-

butions are non-stationary. In the first model, we shall consider the query generator to be “state-driven”. That is, the query generator *itself* will be considered to be making transitions within the states of a finite Markov chain. The pioneering endeavor to solve this type of non-stationarity was suggested by Lam *et. al.* in [48]. In the model defined by Lam *et. al.*, there is one state associated with each record of the list and thus, in each state, one specific record of the linear list is assumed to be accessed. We extend this model by generalizing the states and allowing stochastic accesses to each of the elements of the list, and that, in every state of the query generator. In other words, the distribution associated with each state need not be the unit vector. We shall call this model the Markovian Switching Query Generator (MSQG) and it is analogous to the switching environments, as described in [66, 90]. This case is analyzed in Section 6.2, for the case when the list organizing rule is MTF.

In Section 6.3, in a more daring attempt to tackle the problem, we make the model for the query generator to be far more general. In this case, we permit the underlying distribution $\mathcal{S}(n)$, at the time instant n to be a random vector in itself, and this vector is drawn from some unknown underlying distribution. This model is called the Random Query Generation (RQG) Model.

In this chapter, we shall analyze the MTF rule for the MSQG and RQG models. The techniques that are used for solving these two problems are distinct. In the MSQG model, one possible solution is to transform the non-homogeneous chain into a homogeneous Markov chain based on a product space associated with the states of the query generator and the permutations of the list. Although this approach is theoretically tractable, and would lead to a solution analogous to the one in [90], in reality, the problem is intractable because of the magnitude of the product space. The technique used for the RQG model will also be explained in greater detail in the appropriate sections. We also present experimental results which support our analytical derivations.

6.2 Markovian Switching Query Generators

In this section we shall present our first model for non-stationary query distributions. In this approach, we associate the set of states $\{Q_i | 1 \leq i \leq e\}$ with the non-stationary query generator. Furthermore, when the query generator is in the state Q_i , it uses the probability distribution $S_i = [s_{i1}, s_{i2}, \dots, s_{iN}]^T$, for generating queries. The components of S_i obey the following constraint for all $1 \leq i \leq e$:

$$\begin{aligned} 0 \leq s_{ik} &\leq 1, \quad \text{for } 1 \leq k \leq N. \\ \sum_{k=1}^N s_{ik} &= 1. \end{aligned}$$

We shall assume that the query generator makes transitions between the set of states $\{Q_1, Q_2, \dots, Q_e\}$ in a Markovian fashion and hence it will itself be modeled as a Markov Chain. As mentioned earlier, although this modeling approach is directly based on the work of Tsetlin in [90], the technique by which we endeavor to solve the problem is distinct from his.

The situation described above corresponds to one of cascading two Markov chains. The query generator forms the first Markov Chain and the set of permutations are the states of the second Markov Chain. The outputs from the first chain effect the transitions made in the second chain. Two approaches to the analysis of cascaded chains can be seen. To present the problem in the correct perspective, we start with the "intuitive" approach to analyze the cascading of these two chains, which unfortunately is not valid.

In the "intuitive" approach to solve the above problem, we try to solve each of the Markov chains separately, as follows. The first step involves calculating the equilibrium probabilities of the first chain (i.e. the one associated with the query generator). Based on these, we calculate the steady state query distribution $\bar{S}$ which is observable as the "output" of this chain obtained using the law of total probability. With this information about $\bar{S}$ at hand, the steady state probabilities of the list being in one of its $N!$ arrangements can be easily written down, when the reorganizing heuristic is either the MTF or TR. The problems with this approach are

similar to those encountered in [90] in which the equilibrium distribution is shown to be function not only of $\bar{S}$ but also of the mean time between switching operations. Indeed, the result that the complete characterization of the MSQG (in the the cross product of the rule-list space) affects the expected cost of retrieval in a linear list, has been shown to be true by Lam *et. al.* [48]. This will be clarified presently.

6.2.1 Exact Analysis Using the Composite Chain

6.2.1.1 Asymptotic Analysis for the Query Generation System

An exact analysis of the MTF rule would require that we construct a composite Markov chain, in which the resultant state space is the cross product of the states of the list, and the states of the query generator. The set of states of the list is simply the set of all permutations $\Pi = \{\pi_i \mid i = 1, 2, \dots, N!\}$. The query generator Q can be assumed to be in one of the states of the set $\{Q_1, Q_2, \dots, Q_c\}$. We shall denote by $Q(n)$ the state of the query generator at the time instant n . The transition matrix for the Markov Chain representing the query generator is $\Delta = [\delta_{ij}]$, where, $\delta_{ij} = Pr(Q(n+1) = Q_j \mid Q(n) = Q_i)$. Also, when the query generator is in the state Q_i , it uses the probability distribution $\mathcal{S}_i$ to generate accesses to the elements of the linear list, where $\mathcal{S}_i = [s_{i1}, s_{i2}, \dots, s_{iN}]^T$.

We assume that the Markov Chain of the MSQG is ergodic. An understanding of the concepts introduced here leads to a trivial solution in the case when the chain is absorbing. Since the chain will be assumed to be ergodic, it will possess a unique stationary distribution, which is independent of the initial state occupancy probabilities. We shall denote the state occupancy probability vector at the time instant n by $\mathbf{P}(n) = [p_1(n), p_2(n), \dots, p_c(n)]^T$ and the asymptotic value of $\mathbf{P}(n)$ by $\mathbf{P}^* = [p_1^*, p_2^*, \dots, p_c^*]^T$ which satisfies the following equation [66]:

$$\Delta^T \cdot \mathbf{P}^* = \mathbf{P}^*. \quad (6.2)$$

It is well known that the eigenvalues of the matrix Δ determine the rate of convergence of the probability $\mathbf{P}(n)$. Let us denote the set of eigenvalues of Δ by $\{\lambda_1, \lambda_2, \dots, \lambda_c\}$ and let these be in descending order of their magnitudes. Clearly,

$\lambda_1 = 1$ since unity must be an eigenvalue of every stochastic matrix. The eigenvectors of Δ are given by $\{\underline{v}_m\}$, where $\underline{v}_m$ corresponds to the eigenvalue λ_m . It is shown in [66] that closed form expressions for $\mathbf{P}(n)$ can be written down as:

$$\mathbf{P}(n) = \sum_{m=1}^c a_m \lambda_m^n \underline{v}_m. \quad (6.3)$$

where, a_m 's are constants depending on $\mathbf{P}(0)$. Additionally, we know that $\underline{v}_1 = \mathbf{P}^*$, since $\lambda_1 = a_1 = 1$.

We can use the expression in (6.3) to analyze the behavior of the MSQG as a function of time. The total probability of accessing the record R_i at the time instant n can be written down as:

$$\begin{aligned} s_i(n) &= \text{Pr}(R_i \text{ is accessed at time } n) \\ &= \sum_{m=1}^c \text{Pr}(R_i \text{ is accessed} \mid Q(n) = Q_m) \cdot \text{Pr}(Q(n) = Q_m) \\ &= \sum_{m=1}^c s_{mi} \cdot \text{Pr}(Q(n) = Q_m) \\ &= \sum_{m=1}^c s_{mi} \cdot p_m(n). \end{aligned} \quad (6.4)$$

Using (6.3), it is easy to see that $s_i(n)$ can be expressed in the form:

$$s_i(n) = \bar{s}_i + \sum_{m=2}^c c_m \lambda_m^n. \quad (6.5)$$

where, $\bar{s}_i = \sum_{l=1}^c s_{li} \cdot p_l^*$ and is the asymptotic value of $s_i(n)$.

With this insight, the asymptotic expected query distribution (when the MSQG has converged) is obtained using the law of total probability as:

$$\bar{\mathcal{S}} = \sum_{i=1}^c p_i^* \mathcal{S}_i. \quad (6.6)$$

This implies that the j^{th} component of $\bar{\mathcal{S}}$ is the weighted combination of the j^{th} components of the vectors $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_c\}$.

6.2.1.2 Analysis of the Composite Chain

Having defined the characteristics of the query generator, we now turn our attention to the composite chain mentioned in the preamble of Section 6.2.1.1. Each state in the composite Markov chain is an ordered pair of the form $\langle \pi_i, Q_m \rangle$. Let us denote the state of the composite Markov Chain at the time instant n by $\phi(n)$. Then $\phi(n)$ is simply the ordered pair $\langle L(n), Q(n) \rangle$, where $L(n)$ and $Q(n)$ represent the state of the list and the query generator at time instant n . The transition probabilities of the composite Markov Chain are derived according to the equation:

$$\begin{aligned}
 Pr(\phi(n+1) = \langle \pi_i, Q_m \rangle | \phi(n) = \langle \pi_k, Q_l \rangle) \\
 &= Pr(L(n+1) = \pi_i, Q(n+1) = Q_m | \phi(n) = \langle \pi_k, Q_l \rangle), \\
 &= Pr(L(n+1) = \pi_i | \phi(n) = \langle \pi_k, Q_l \rangle, Q(n+1) = Q_m) \\
 &\quad \cdot Pr(Q(n+1) = Q_m | \phi(n) = \langle \pi_k, Q_l \rangle) \quad (\text{using the chain rule})
 \end{aligned} \tag{6.7}$$

which reduces to the following:

$$\begin{aligned}
 &= Pr(L(n+1) = \pi_i | L(n) = \pi_k, Q(n) = Q_l) \\
 &\quad \cdot Pr(Q(n+1) = Q_m | Q(n) = Q_l) \quad (\text{dropping extraneous conditions}) \\
 &= Pr(L(n+1) = \pi_i | L(n) = \pi_k, Q(n) = Q_l) \cdot \delta_{lm}.
 \end{aligned} \tag{6.8}$$

For the general case, when the list to be ordered has N elements, and the environment has e states, the composite chain has $e \cdot N!$ states. If the resulting Markov Chain is ergodic, the problem is “simply” one of computing *its* stationary distribution. Let $Pr^*(\langle \pi_i, Q_m \rangle)$ be the equilibrium (stationary) probability of the composite chain being in the state $\langle \pi_i, Q_m \rangle$. Once these probabilities are known, the expected retrieval cost can be easily computed from the following generalization of (6.1):

$$C_\tau = \sum_{m=1}^e \sum_{i=1}^{N!} Pr^*(\langle \pi_i, Q_m \rangle) \cdot \text{Cost}(\pi_i, Q_m). \tag{6.9}$$

where $\text{Cost}(\pi_i, Q_m)$ is the expected cost associated with the permutation π_i , if it was interacting with the environment Q_m .

Although (6.9) is the exact formula for computing the cost, its direct use is, of course, cumbersome, due to the large number ($e \cdot N!$) of stationary probabilities that

must be computed. Thus even though this approach is analytically sound, we would prefer an alternate method to compute C_r . It can be shown that the expected cost for *any* linear list organizing heuristic can be written down as [48]:

$$C_r = \sum_{j=1}^N \bar{s}_j \left\{ \sum_{\substack{i=1,2,\dots,N \\ i \neq j}} (1 + b(i, j|j)) \right\}. \quad (6.10)$$

where $\bar{s}_j$ denotes the asymptotic probability of accessing element R_j and $b(i, j|j)$ represents the probability that the element R_i precedes the element R_j , given that element R_j has been accessed. Throughout this chapter (as in all the current literature) we opt to compute C_r using (6.10) as opposed to (6.9).

We shall now present our first result for the MSQG system in which $N = 2$ and the number of environments is arbitrary.

Theorem 6.1 *Consider the case of MSQG in which the list being organized has two elements ($N = 2$) and the query generator can be in one of e states, governed by the transition matrix $\Delta = [\delta_{ij}]$. Then, $Pr^*(\langle ij \rangle)$, the asymptotic probability that the list is in the state $\langle ij \rangle$, is given by:*

$$Pr^*(\langle ij \rangle) = \sum_{m=1}^e p_m^* s_{mi}.$$

Proof:

Let $w_{ij;m}(n)$ denote the probability that the composite chain is in the state $\langle ij; Q_m \rangle$, at the time instant n , where $i, j = 1, 2$ and $i \neq j$. Then, using (6.8), the transition probabilities in the composite Markov chain can be written down as:

$$Pr(\phi(n+1) = \langle ij; Q_m \rangle \mid \phi(n) = \langle uv; Q_w \rangle) = s_{wi} \delta_{wm}.$$

In this scenario, the typical transitions in the composite Markov Chain are shown in Figure 6.2. From an examination of the composite Markov chain in Figure 6.2 we can write the following equations:

$$\begin{aligned} w_{ij;m}(n+1) &= \sum_{l=1}^e \{w_{ij;l}(n) s_{li} \delta_{lm}\} + \sum_{l=1}^e \{w_{ji;l}(n) s_{li} \delta_{lm}\} \\ &= \sum_{l=1}^e s_{li} \{w_{ij;l}(n) + w_{ji;l}(n)\} \delta_{lm}. \end{aligned} \quad (6.11)$$

Figure 6.2: This figure depicts typical transitions in the composite Markov chain for the system in which the list $\mathcal{R}$ has 2 elements. In the illustration, the query generator switches between two states i and j .

Using (6.11) and the laws of total probability, we can compute the total probability that the list will be in the order $\langle ij \rangle$ at the time instant $n + 1$ as:

$$\begin{aligned}
 Pr(L(n+1) = \langle ij \rangle) &= \sum_{m=1}^e w_{ij;m}(n+1) \\
 &= \sum_{m=1}^e \sum_{l=1}^e s_{li} \{w_{ij;l}(n) + w_{ji;l}(n)\} \delta_{lm} \\
 &\quad \text{(by interchanging the order of summations)} \\
 Pr(L(n+1) = \langle ij \rangle) &= \sum_{l=1}^e \sum_{m=1}^e s_{li} \{w_{ij;l}(n) + w_{ji;l}(n)\} \delta_{lm} \\
 &= \sum_{l=1}^e s_{li} \{w_{ij;l}(n) + w_{ji;l}(n)\} \left(\sum_{m=1}^e \delta_{lm} \right) \\
 \text{and since } \sum_{m=1}^e \delta_{lm} &= 1, \text{ we get} \\
 Pr(L(n+1) = \langle ij \rangle) &= \sum_{l=1}^e s_{li} \{w_{ij;l}(n) + w_{ji;l}(n)\} \tag{6.12}
 \end{aligned}$$

Consider the term $w_{ij;l}(n) + w_{ji;l}(n)$. Since this term is independent of the ordering of the elements of the list, and only involves l , the index of the environment, we have,

$$w_{ij;l}(n) + w_{ji;l}(n) = Pr(Q(n) = Q_l). \tag{6.13}$$

Using (6.13), (6.12) can be rewritten as:

$$Pr(L(n+1) = \langle ij \rangle) = \sum_{l=1}^e s_{li} Pr(Q(n) = Q_l) \tag{6.14}$$

Since the Markov chain representing the query generator be ergodic, $\lim_{n \rightarrow \infty} Pr(Q(n) = Q_l)$ exists as given by (6.2). Thus, we conclude that,

$$Pr^*(\langle ij \rangle) = \lim_{n \rightarrow \infty} Pr(L(n) = \langle ij \rangle) = \sum_{l=1}^e p_l^* s_{li}.$$

which proves the result. $\square$

Even though the number of elements N is just 2, the computation of the asymptotic probabilities of the composite chain is quite involved. It is coincidental that even though this computation is elaborate, it can be simplified using (6.13). However the insight we get from this analysis is interesting because it presents us with

a technique by which the asymptotic probabilities of the individual chains can be coalesced to give us the asymptotic properties of the composite chain. Interestingly, the *actual* elements of Δ , the transition probability matrix of the MSQG, do not explicitly appear in the asymptotic probabilities; they are implicitly included by way of the asymptotic probabilities $\{p_i^*\}$.

It is clear that the above techniques can be used if the number of records is $N > 2$. In this case, as mentioned earlier, the composite Markov chain has $\epsilon \cdot N!$ states and evaluating the eigenvectors of the $(\epsilon \cdot N!) \times (\epsilon \cdot N!)$ matrix, the asymptotic probabilities of the resulting Markov Chain can be derived. However this strategy is ludicrous even for values of $N \geq 4$, and so we will have to search for an alternate strategy. In order to do so, rather than concentrate on the entire list, we will focus our attention on the asymptotic probability of an element R_i preceding another element R_j . This, of course, is quite in step with the traditional concepts involving stationary environments.

It is appropriate to point out the difference between our approach and the approach adopted in [48] to find the conditional probabilities of the form $b(i, j|j)$. Lam *et. al.* use the properties of the reversed-time chain corresponding to the query generator (such as its mean time for first passage), to compute the asymptotic values of the conditional probabilities mentioned above. Although these computations are elegant, the implicit assumption made in this analysis is that the reversed-time chain is homogeneous. This assumption is valid only after the forward chain has converged. Note also that the transition probabilities for the reversed-time chain are computed in terms of the transition probabilities of the original chain, and the asymptotic distribution of the forward chain. In other words, Lam's approach cannot model the transient behavior of the time-varying chain. In contrast, in our approach, we construct a difference equation involving these probabilities. As a by-product, we derive the asymptotic value for these probabilities as well. Because of the assumption needed to solve these difference equations, our asymptotic values do not exactly agree with those derived by Lam *et. al.* [48].

If the number of elements in the list, N is arbitrary, the composite chain has

$e \cdot N!$ states. In this case, we would like to study the lumped form of the composite chain [40]. In the lumped chain, the state $\langle ij; m \rangle$ denotes the state in which the record R_i precedes record R_j and the query generator is in the state Q_m . We denote the event “ R_i precedes R_j ” as $R_i \prec R_j$. If ${}_iP_j(n)$ denotes the probability that record R_i precedes R_j at the time instant n , then, proceeding exactly as in the proof of Theorem 6.1, we can deduce that:

$${}_iP_j(n+1) = \sum_{l=1}^e s_{li} Pr(Q(n) = Q_l) + \sum_{l=1}^e (1 - s_{li} - s_{lj}) w_{ij;l}(n). \quad (6.15)$$

(6.15) is a recurrence relation involving ${}_iP_j(n+1)$ and the state occupancy probabilities of the composite chain, i.e., $w_{ij;l}(n)$. To solve this recurrence relation, we need to make the simplifying assumption about the transient behavior of the composite chain:

Pseudo-independence Assumption:

The composite chain, determined by the probabilities $w_{ij;l}(n)$, is said to be pseudo-independent if:

$$w_{ij;l}(n) = {}_iP_j(n) Pr(Q(n) = Q_l). \quad (6.16)$$

Note that (6.16) **does not** imply that the list configurations and the events denoting query generator states are statistically independent. Indeed statistical independence between these events is a much more stringent requirement which would impose the following conditions:

$$Pr(\langle \pi_k, Q_l \rangle)(n) = Pr(\pi_k)(n) Pr(Q(n) = Q_l) \quad 1 \leq k \leq N!, \quad 1 \leq l \leq e. \quad (6.17)$$

Thus whereas (6.16) states that:

$$\sum_{\pi_k | R_i \prec R_j} Pr(\langle \pi_k, Q_l \rangle)(n) = Pr(Q(n) = Q_l) \sum_{\pi_k | R_i \prec R_j} Pr(\pi_k)(n) \quad (6.18)$$

(6.17) imposes equality of individual terms of the summations on the LHS and RHS of (6.18). It is in this sense that Pseudo-independence is a much weaker constraint than independence itself¹.

¹Observe that (6.16) leads to a set of $e \cdot N(N-1)$ linear equations, from which it is not possible to uniquely “solve” for $e \cdot N!$ unknowns of the form $Pr(\langle \pi_i, Q_k \rangle)$. Thus independence cannot be inferred from (6.16).

It is easy to show that under this assumption, the asymptotic probability $b(i, j|j)$ is the same as $b(i, j)$. Hence, we can concentrate on computing the latter quantity. We now state and prove a general result concerning the asymptotic probability of record R_i preceding R_j in the case of the MSQG model, when the number of records N is arbitrary.

Theorem 6.2 *Let ${}_iP_j(n)$ denote the probability that record R_i precedes R_j at the time instant n , whose asymptotic value is ${}_iP_j^*$. Also let S be the mean asymptotic query distribution defined by (6.6). Then, under the assumption of Pseudo-independence, the following results hold:*

1. ${}_iP_j(n+1) = \left(\prod_{k=1}^n \{1 - s_i(k-1) - s_j(k-1)\} \right) \cdot {}_iP_j(0) + \sum_{k=1}^n s_i(k-1) \left(\prod_{l=k+1}^n \{1 - s_i(l-1) - s_j(l-1)\} \right).$
2. ${}_iP_j^* = \lim_{n \rightarrow \infty} {}_iP_j(n) = \frac{\bar{s}_i}{\bar{s}_i + \bar{s}_j}.$

Proof:

By the nature of the heuristic used, we can write down the recursive form for the distribution of the *total* probabilities ${}_iP_j(n+1)$ as:

$${}_iP_j(n+1) = \begin{cases} 1 & \text{if record } R_i \text{ is accessed at time } n \\ 0 & \text{if record } R_j \text{ is accessed at time } n \\ {}_iP_j(n) & \text{if some record other than } R_i \text{ or } R_j \\ & \text{was accessed at time } n \end{cases} \quad (6.19)$$

We note that the record R_i is accessed at the time instant n , with probability $s_i(n)$. Also the probability that some record other than R_i or R_j is accessed at time instant n is simply given by $1 - s_i(n) - s_j(n)$. The probabilities mentioned above can be rewritten in the form²:

$${}_iP_j(n+1) = \begin{cases} 1 & \text{w.p. } s_i(n) \\ 0 & \text{w.p. } s_j(n) \\ {}_iP_j(n) & \text{w.p. } 1 - s_i(n) - s_j(n). \end{cases} \quad (6.20)$$

---

²w.p. denotes "with probability"

Hence, we can write down the total probability ${}_iP_j(n+1)$ as follows:

$$\begin{aligned} {}_iP_j(n+1) &= 1 \cdot s_i(n) + 0 \cdot s_j(n) + {}_iP_j(n) \cdot (1 - s_i(n) - s_j(n)) \\ &= s_i(n) + {}_iP_j(n) \cdot (1 - s_i(n) - s_j(n)). \end{aligned} \quad (6.21)$$

This is a difference (or recurrence) equation, whose solution yields the required probability of the form ${}_iP_j(n)^3$. Since the coefficients are not constants, the difference equation is not trivially solved and we shall tackle this presently.

For the ease of the derivation, we now introduce some simplifying notation:

$$\begin{aligned} x_n &\triangleq {}_iP_j(n) \\ b_{n+1} &\triangleq 1 - s_i(n) - s_j(n) \\ c_{n+1} &\triangleq s_i(n). \end{aligned}$$

With this notation, the difference equation (6.21) can be rewritten as:

$$x_{n+1} = b_{n+1}x_n + c_{n+1}. \quad (6.22)$$

Consider the following transformation of variables.

$$\begin{aligned} x_0 &= y_0 \\ x_n &= \left(\prod_{k=1}^n b_k \right) \cdot y_n \text{ for } n > 0 \end{aligned} \quad (6.23)$$

Using this substitution, (6.22) becomes:

$$\left(\prod_{k=1}^{n+1} b_k \right) \cdot y_{n+1} = b_{n+1} \cdot \left(\prod_{k=1}^n b_k \right) \cdot y_n + c_{n+1}.$$

If we divide both sides of this equation by $\left(\prod_{k=1}^{n+1} b_k \right)$, we get:

$$y_{n+1} = y_n + d_{n+1} \quad (6.24)$$

where

$$d_{n+1} = c_{n+1} / \prod_{k=1}^{n+1} b_k$$

It is now obvious that the difference equation (6.24) has the simple solution:

---

³If the query distribution were to be stationary, we would have $s_i(n) = s_i$ for all n . In this case, the difference equation simply reduces to one with constant coefficients. The solution method is straightforward, and it can be easily shown that for this special case, $\lim_{n \rightarrow \infty} {}_iP_j(n) = s_i / (s_i + s_j)$. This clearly agrees with the result derived in [81].

$$y_n = y_0 + \sum_{j=1}^n d_j \quad (6.25)$$

Hence, by transforming the solution in (6.25) back to the original variables of our problem, we get the general solution of the original difference equation (6.21) as:

$$\begin{aligned} x_n &= \left(\prod_{k=1}^n b_k \right) \cdot x_0 + \left(\prod_{k=1}^n b_k \right) \cdot \left\{ \sum_{l=1}^n \frac{c_l}{\prod_{k=1}^l b_k} \right\} \\ &= \left(\prod_{k=1}^n b_k \right) \cdot x_0 + \sum_{k=1}^n c_k \left(\prod_{l=k+1}^n b_l \right). \end{aligned} \quad (6.26)$$

In terms of the original variables of our problem, we rewrite (6.26) as:

$$\begin{aligned} {}_iP_j(n+1) &= \left(\prod_{k=1}^n \{1 - s_i(k-1) - s_j(k-1)\} \right) \cdot {}_iP_j(0) + \\ &\quad + \sum_{k=1}^n s_i(k-1) \left(\prod_{l=k+1}^n \{1 - s_i(l-1) - s_j(l-1)\} \right). \end{aligned} \quad (6.27)$$

and the first result of the theorem is proved.

Although (6.27) gives the general expression for the probability that a record R_i precedes another record R_j at a given instant n , it is inconvenient to use this to derive the asymptotic value of the probability. In fact, we first need to prove the *existence* of the asymptotic value. Since the Markov chain for the MSQG is ergodic, we know that $|\lambda_2| < 1$, and hence $\lambda_2^n \rightarrow 0$, as $n \rightarrow \infty$. Our aim now is to find upper and lower bounds on the values of ${}_iP_j(n)$. These bounds can assist in the determination of the asymptotic value of ${}_iP_j(n)$. Now, it is easy to see that there exists a sufficiently large value n_0 such that:

$$\begin{aligned} s_i(n) &= \bar{s}_i + \sum_{m=2}^e c_m \lambda_m^n \\ &\leq \bar{s}_i + e|c_{max}| \cdot |\lambda_2|^{n_0} \quad \forall n > n_0. \end{aligned} \quad (6.28)$$

Similarly,

$$s_i(n) \geq \bar{s}_i - e|c_{max}| \cdot |\lambda_2|^{n_0} \quad \forall n > n_0. \quad (6.29)$$

Using (6.28) and (6.29), we can bound the quantity $s_i(n) + s_j(n)$ as:

$$\begin{aligned} \bar{s}_i + \bar{s}_j - 2\epsilon|c_{max}| \cdot |\lambda_2|^{n_0} \\ \leq s_i(n) + s_j(n) \\ \bar{s}_i + \bar{s}_j + 2\epsilon|c_{max}| \cdot |\lambda_2|^{n_0} \quad \forall n > n_0. \end{aligned}$$

Or alternately,

$$\begin{aligned} 1 - \bar{s}_i - \bar{s}_j - 2\epsilon|c_{max}| \cdot |\lambda_2|^{n_0} \\ \leq 1 - s_i(n) - s_j(n) \leq 1 - \bar{s}_i - \bar{s}_j + 2\epsilon|c_{max}| \cdot |\lambda_2|^{n_0} \quad \forall n > n_0. \end{aligned}$$

Consider the two new difference equations involving the quantities ${}_i\bar{P}_j(n)$ and ${}_i\underline{P}_j(n)$, which are intended to represent an upper bound and lower bound on ${}_iP_j$ respectively. These difference equations are defined only when $n > n_0$, with $c = \epsilon \cdot |c_{max}|$.

$${}_i\bar{P}_j(n+1) = (\bar{s}_i + c|\lambda_2|^{n_0}) + (1 - \bar{s}_i - \bar{s}_j + 2c|\lambda_2|^{n_0}) \cdot {}_i\bar{P}_j(n) \quad (6.30)$$

$${}_i\underline{P}_j(n+1) = (\bar{s}_i - c|\lambda_2|^{n_0}) + (1 - \bar{s}_i - \bar{s}_j - 2c|\lambda_2|^{n_0}) \cdot {}_i\underline{P}_j(n) \quad (6.31)$$

It is easy to verify that

$${}_i\bar{P}_j(n) \geq {}_iP_j(n) \geq {}_i\underline{P}_j(n) \quad \text{for all } n \geq n_0. \quad (6.32)$$

Now (6.30) and (6.31) are constant coefficient difference equations, with asymptotic values given by:

$$\lim_{n \rightarrow \infty} {}_i\bar{P}_j(n) = \frac{\bar{s}_i + c|\lambda_2|^{n_0}}{\bar{s}_i + \bar{s}_j - 2c|\lambda_2|^{n_0}} \quad (6.33)$$

$$\lim_{n \rightarrow \infty} {}_i\underline{P}_j(n) = \frac{\bar{s}_i - c|\lambda_2|^{n_0}}{\bar{s}_i + \bar{s}_j + 2c|\lambda_2|^{n_0}} \quad (6.34)$$

Note that ${}_i\bar{P}_j(n)$ and ${}_i\underline{P}_j(n)$ bound ${}_iP_j(n)$ from above and below respectively. Moreover, we have derived the asymptotic values of ${}_i\bar{P}_j(n)$ and ${}_i\underline{P}_j(n)$ as $n \rightarrow \infty$. Hence if the quantity ${}_iP_j(n)$ converged as $n \rightarrow \infty$, its limit must be bounded by the asymptotic values of ${}_i\bar{P}_j$ and ${}_i\underline{P}_j$. These asymptotic values clearly depend on the choice of n_0 . Since the choice of n_0 is arbitrary, we can choose it to be as large

as we please. An examination of (6.33) and (6.34) reveals that as $n_0 \rightarrow \infty$, both these bounds converge to a **common** value, which, by (6.32) must also be the limit of ${}_iP_j(n)$ as $n \rightarrow \infty$. Hence we have established that:

$${}_iP_j^* = \frac{\bar{s}_i}{\bar{s}_i + \bar{s}_j}.$$

which is the second claim of the theorem. □

Remark:

It must be noticed that the result derived in Theorem 6.1 is a special case of Theorem 6.2, when the list has exactly two elements (i.e., $N = 2$). That the expressions given by Theorem 6.1 and Theorem 6.2 are identical, can be seen by observing that $\bar{s}_i + \bar{s}_j = 1$, if the list has exactly two elements. Since the MTF and TR reorganize the list in the same manner if $N = 2$, the result derived in Theorem 6.1 applies to the TR heuristic as well. However, if $N > 2$, Theorem 6.2 only applies to MTF.

6.3 Random Query Generator

In the previous section, we had demonstrated the analysis technique in the case when the query generator was itself modeled as a Markov chain. In other words, we analyzed the MTF rule in the case when the probability distribution vector $\mathcal{S}$ switched between a set of e vectors. Under this model, the asymptotic value of the probability ${}_iP_j$ was derived.

In this section, we explore generalizations of the above model of query generation. That is, we assume no knowledge of the mechanism by which the query generator is selecting its distributions. The query generation is completely arbitrary and is described by a random *process*. In particular, we no longer require that the probability distributions selected by the query generator be restricted to a finite set and thus, we allow the probability distribution vector $\mathcal{S}(n) = [s_1(n), s_2(n), \dots, s_N(n)]^T$ to itself be a *random* vector, possessing a distribution $f(\mathcal{S}(n))$. The mean of $\mathcal{S}(n)$ will be denoted by $\mu(n) = [\mu_1(n), \mu_2(n), \dots, \mu_N(n)]^T$.

A good deal of insight into this model of query generation can be obtained from the following explanation. Let us consider the computation of the *total probability* that an element R_i of the list is accessed. Now,

$$Pr(R_i \text{ is accessed at time } n | \mathcal{S}(n)) = s_i(n). \quad (6.35)$$

Hence, taking expectations of both sides of (6.35), we obtain $Pr(R_i \text{ is accessed}) = E[s_i(n)] = \mu_i(n)$. This implies that only the mean of the distribution of $\mathcal{S}(n)$ is needed in the computation of the total access probability of any record. All other characteristics of the distribution $f(\mathcal{S}(n))$ are not relevant for this purpose. With this clarification, two avenues for study are open. In the first one, we assume that although $f(\mathcal{S}(n))$ changes with time, the means of the distribution of $\mathcal{S}(n)$ are time invariant. In the subsequent model, we allow the means themselves to vary with time. These models are now presented in the increasing order of their generalities.

6.3.1 Time Invariant Means

We state our first result in this connection.

Theorem 6.3 *Let $\mathcal{S}(n)$ be the query distribution used by the query generator at the time instant n . If $\mathcal{S}(n)$ has a distribution $f(\mathcal{S}(n))$ with a time invariant mean $\mu = [\mu_1, \mu_2, \dots, \mu_N]^T$, then the following results hold for the MTF heuristic:*

1. *The Resulting Markov chain representing the data restructuring operation is homogeneous.*
2. *The steady state behavior can be obtained by merely solving the homogeneous chain using μ as the hypothetical query distribution.*

Proof:

Consider the fundamental Markov Chain associated with the problem and the MTF heuristic. In this chain, the states of the chain are all the permutations of the list. Let $L(n)$ denote the state of the chain at any time instant n . We proceed to examine the transition probabilities of this chain.

Let $\langle i_1, i_2, \dots, i_N \rangle$ and $\langle j_1, j_2, \dots, j_N \rangle$ denote any two permutations of the integers $\{1, 2, \dots, N\}$. The list is accessed at the time instant n according to the distribution $\mathcal{S}(n) = [s_1(n), s_2(n), \dots, s_N(n)]^T$. Consider the transition from the order $\Pi_1 = \langle R_{j_1}, R_{j_2}, \dots, R_{j_N} \rangle$, to the order $\Pi_2 = \langle R_{i_1}, R_{i_2}, \dots, R_{i_N} \rangle$. In the MTF heuristic, this transition is possible only if R_{i_1} is accessed and one of the following conditions are true:

- (i) $i_m = j_m \quad \forall m = 1, 2, \dots, N$, or,
- (ii) $\exists m$ such that $i_1 = j_m$ and

$$\langle i_2, i_3, \dots, i_N \rangle = \langle j_1, j_2, \dots, j_{m-1}, j_{m+1}, \dots, j_N \rangle.$$

If none of the above conditions are satisfied, the MTF rule cannot produce a transition from the list Π_1 to the list Π_2 . In the following derivation, we shall restrict ourselves to only those combinations of the indices $\langle i_1, i_2, \dots, i_N \rangle$ and $\langle j_1, j_2, \dots, j_N \rangle$ for which the desired transition is possible. This is because of the fact that in cases where the choice of these indices is such that a transition between the list configuration Π_1 to the desired state Π_2 is not possible under the MTF heuristic, this total transition probability simply evaluates to zero. Indeed, when such a transition is possible using the MTF rule, we have,

$$\begin{aligned}
 Pr(L(n+1) = \Pi_2 | L(n) = \Pi_1, \mathcal{S}(n)) \\
 &= Pr(R_{i_1} \text{ is accessed} \mid \text{query distribution is } \mathcal{S}(n)) \\
 &= s_{i_1}(n).
 \end{aligned} \tag{6.36}$$

The total probability of the going from the state Π_1 to the desired state Π_2 (in cases where such a transition is possible) can be written using the laws of total probability and (6.36) as:

$$\begin{aligned}
 Pr(L(n+1) = \Pi_2 | L(n) = \Pi_1) \\
 &= \sum_{\text{all values of } \mathcal{S}(n)} s_{i_1}(n) \cdot f(\mathcal{S}(n)),
 \end{aligned} \tag{6.37}$$

where $f(\mathcal{S}(n))$ is the distribution of $\mathcal{S}(n)$. But the RHS of (6.37) is exactly the formal definition of $E[s_{i_1}(n)] = \mu_{i_1}(n)$. Since the means in our case are time invariant $E[s_{i_1}(n)] = \mu_{i_1}$. Hence,

$$Pr(L(n+1) = \Pi_2 | \Pi_1) = \mu_{11}.$$

Hence *all* the transition probabilities in the Markov chain of the MTF rule become time invariant and thus although the problem *itself* exhibits nonstationarity, the underlying Markov chain is homogeneous. Moreover, the transition probabilities are identical to those of a homogeneous Markov chain obtained if μ was the stationary distribution. This proves the theorem. $\square$

Notice that the derivation did not make any assumptions about the parametric form or the variance of the random vector $S(n)$. Just the restriction of the time invariance of the **mean** of the random vector $S(n)$ was sufficient to ensure the homogeneity of the resulting chain. The reader must also notice that although $S(n)$ itself is a random vector, the total probability that any specific record R_i is accessed, is time invariant. Thus although the result was proved only for the case of the MTF heuristic, the result holds for *any* time invariant list organizing heuristic.

6.3.2 Time Varying Means

Given that time invariant means are relatively easy to analyze, it is natural to investigate the consequences of having the means themselves vary with time. Let us denote the mean distribution used by the query generator at the time instant n be $\mu(n)$. If $\mu(n)$ is allowed to be an arbitrary function of n , the analysis is clearly intractable. Some insight into the problem could be obtained by suitably restricting the variation of $\mu(n)$ with n . We now present our final result based on a family of functions which restricts the variation of $\mu(n)$. The restriction we impose is very weak : all that we require is that the means themselves form a convergent sequence. However the result obtained is very general and includes all the known results involving the MTF rule.

Theorem 6.4 *Let $S(n)$ be the (random) probability vector characterizing the accesses at time instant n which obeys the distribution $f(S(n))$. Further $\mu(n) = E[S(n)]$. If additionally the following condition holds:*

$$\lim_{n \rightarrow \infty} \mu(n) = \mu^* = [\mu_1^*, \mu_2^*, \dots, \mu_N^*]^T$$

then, under the MTF heuristic, the asymptotic probability, ${}_iP_j^*$ that record R_i precedes record R_j is given by:

$${}_iP_j^* = \frac{\mu_i^*}{\mu_i^* + \mu_j^*}.$$

Proof:

As before, we denote the probability that the record R_i precedes the record R_j at the time instant n by ${}_iP_j(n)$. We proceed in a manner analogous to that adopted for the case of the analysis of the MSQG model. In other words, we first construct a difference equation and examine its properties.

Now, under the MTF heuristic, the distribution of the *total* probability can be written in a recursive form, as follows:

$${}_iP_j(n+1) = \begin{cases} 1 & \text{w.p. } s_i(n) \\ 0 & \text{w.p. } s_j(n) \\ {}_iP_j(n) & \text{w.p. } 1 - s_i(n) - s_j(n). \end{cases} \quad (6.38)$$

Based on this equation, we can write down the *conditional expected value* of ${}_iP_j(n+1)$ as:

$$E[{}_iP_j(n+1)|{}_iP_j(n), \mathcal{S}(n)] = s_i(n) + (1 - s_i(n) - s_j(n)){}_iP_j(n). \quad (6.39)$$

In order to get $E[{}_iP_j(n+1)|\mathcal{S}(n)]$, we need to average the quantity $E[{}_iP_j(n+1)|{}_iP_j(n), \mathcal{S}(n)]$ using the distribution of ${}_iP_j(n)$. In doing so, we must note that the terms on the RHS of (6.39) involving components of $\mathcal{S}(n)$ are effectively constants. Hence, taking expectations of both sides of (6.39) a second time, we get,

$$E[{}_iP_j(n+1)|\mathcal{S}(n)] = s_i(n) + (1 - s_i(n) - s_j(n))E[{}_iP_j(n)]. \quad (6.40)$$

In order to get rid of the conditioning on the probability distribution $\mathcal{S}(n)$, we need to take expectations again. But the argument is a little intricate and is as follows.

The expected value of the LHS of (6.40) is $E[E[{}_iP_j(n+1)|\mathcal{S}(n)]] = E[{}_iP_j(n+1)]$. However the terms on the RHS of (6.40) need careful examination. The term

$E[{}_iP_j(n)]$ only depends on the time instant $n - 1$ and the past history. Hence, in averaging w.r.t. the distribution of $\mathcal{S}(n)$, this quantity can effectively be treated as a constant. This necessitates the use of the expectation operator one more time, and on applying the expectation operator for third time, we get,

$$\begin{aligned} E[{}_iP_j(n+1)] &= E[s_i(n)] + (1 - E[s_i(n)] - E[s_j(n)])E[{}_iP_j(n)] \\ &= \mu_i(n) + (1 - \mu_i(n) - \mu_j(n))E[{}_iP_j(n)]. \end{aligned} \quad (6.41)$$

Now the term $E[{}_iP_j(n)]$ simply represents the total probability that record R_i precedes record R_j at the time instant n . With this understanding, (6.41) can be rewritten as:

$${}_iP_j(n+1) = \mu_i(n) + (1 - \mu_i(n) - \mu_j(n)){}_iP_j(n). \quad (6.42)$$

We now wish to find the asymptotic value of ${}_iP_j(n)$ as $n \rightarrow \infty$. To derive this limit, we make use of the result that the mean $\mu(n)$ converges to μ^* as $n \rightarrow \infty$. Now, by the definition of limit, we have,

$$\forall \epsilon > 0 \quad \exists n_0 < \infty \quad \text{such that} \quad \forall n > n_0 \quad \|\mu(n) - \mu^*\| < \epsilon. \quad (6.43)$$

From (6.43), we deduce that the following condition holds whenever $n > n_0$:

$$|\mu_k(n) - \mu_k^*| < \epsilon/\sqrt{N}, \quad k = 1, 2, \dots, N.$$

Equivalently,

$$\mu_k^* - \epsilon/\sqrt{N} < \mu_k(n) < \mu_k^* + \epsilon/\sqrt{N} \quad \forall n > n_0, \text{ and } k = 1, 2, \dots, N.$$

As in the proof of Theorem 6.2, let ${}_i\bar{P}_j(n)$ and ${}_i\underline{P}_j(n)$ denote upper and lower bounds on ${}_iP_j(n)$, respectively. Therefore the values of ${}_iP_j(n)$ derived from (6.42) for all values of $n \geq n_0$ are bounded from above and below by the solutions of the difference equations:

$${}_i\bar{P}_j(n+1) = (\mu_i^* + \epsilon/\sqrt{N}) + (1 - \mu_i^* - \mu_j^* + 2\epsilon/\sqrt{N}) \cdot {}_i\bar{P}_j(n) \quad (6.44)$$

$${}_i\underline{P}_j(n+1) = (\mu_i^* - \epsilon/\sqrt{N}) + (1 - \mu_i^* - \mu_j^* - 2\epsilon/\sqrt{N}) \cdot {}_i\underline{P}_j(n). \quad (6.45)$$

Notice that (6.44) and (6.45) are constant coefficient difference equations, whose asymptotic behavior is easy to analyze. Following the steps identical to those involved in the latter part of the proof of Theorem 6.2 (omitted here in the interest of brevity) we obtain the desired result that:

$$P_j^* = \frac{\mu_i^*}{\mu_i^* + \mu_j^*}.$$

Hence the theorem. □

6.4 Experimental Results

To validate the theoretical results derived in this chapter, various experiments were conducted. Some typical results are reported here.

In the first set of experiments which we report, the intention was to verify the results for the case of the MSQG model. The number of states in the Markov Chain was fixed to be 3, for all experiments. The matrix of transition probabilities used in our experiments is:

$$\Delta = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.7 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}$$

First, the number of elements in the list N was selected. Three **random** probability vectors (of dimension $N \times 1$) $\{\mathcal{S}_i | i = 1, 2, 3\}$ were generated for each experiment. When the query generator was in the state Q_i , the probability vector $\mathcal{S}_i$ was used for generating queries. Based on these probabilities and the steady-state probabilities of the chain characterized by Δ , theoretical values for the asymptotic values of the probabilities P_j^* were obtained. These probabilities were used to calculate the expected retrieval cost, based on (6.10). This quantity has been reported in Table 6.1.

To verify the accuracy of this cost, 50 experiments were run, each consisting of 10,000 time units (where one user query was generated per time unit). The retrieval cost was counted only during the last 5000 time units of each of these experiments.

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

The ensemble average of the 50 time averages so obtained have also been reported in Table 6.1. In Table 6.1, we have reported results for the cases in which the number of records in the list was 5, 10, 15, and 20. In all the cases, we observe that the experimentally observed cost is very close to the theoretical cost. Thus, for example, the percentage difference between the theoretical and the observed values is only 2.05%, for the case when $N = 15$.

No. of elements in list	Theoretical Avg. Retrieval Cost	Observed Avg. cost	Error (in %)
5	2.2849	2.2421	1.8718
10	2.5015	2.4847	0.6726
15	2.3370	2.2890	2.0539
20	2.2973	2.2903	0.3047

Table 6.1: This table summarizes the results of the experiments conducted with the MSQG model of query generation. The Markov chain for the query generator had 3 states. The probability vectors used in these states were randomly generated. The table reports the theoretical expected asymptotic cost, as well as the experimentally observed retrieval cost, in cases when the number of elements in the list is 5, 10, 15, and 20. The variation in the error rate as a function of the number of records is probably due to the fact that the $\mathcal{S}$ vectors were randomly chosen. This phenomenon may be absent if the $\mathcal{S}$ vectors obeyed a common distribution such as Zipf's law.

In the second set of experiments, our aim was to verify the results presented for the RQG model. In these experiments, queries are generated at time n , based on the vector $\mathcal{S}(n)$ with a distribution $f(\mathcal{S}(n))$, whose mean changed with time. For the sake of brevity, we only report the results for the case when the number of elements in the list was 4. Also, in order to satisfy the conditions of Theorem 6.4, $\mu(n)$ was chosen in such a way that it converged to the limiting value μ^* . To achieve this effect, we opted to generate successive generations of the mean vector $\mu(n)$ according to the rule:

$$\mu(n+1) = F^T \cdot \mu(n). \quad (6.46)$$

The initial value $\mu(0)$ was $[0.4, 0.3, 0.2, 0.1]^T$ and the limiting value μ^* was $[0.3290, 0.2093, 0.2048, 0.2567]^T$.

Once again, 50 experiments were conducted, each involving 10,000 time units as mentioned earlier. The mechanism used for query generation at each time unit was as follows:

1. The value of mean vector $\mu(n)$ at the time instant n was determined according to the (6.46).
2. Let $\mu(n) = [\mu_1(n), \mu_2(n), \dots, \mu_N(n)]^T$. We generate a random vector $S'(n) = [s'_1(n), s'_2(n), \dots, s'_N(n)]^T$, where $s'_i(n)$ a random variable uniformly distributed in the open interval $(0, \mu_i(n))$. To transform $S'(n)$ to be a probability vector, we merely normalize the components of $S'(n)$, so as to sum to unity. This normalized vector is assigned to $S(n)$, and it easy to see that $S(n)$ has the mean $\mu(n)$.
3. Generate a query based on the distribution $S(n)$.

As before, only the last 5000 time units of each experiment were used in computing the average retrieval cost. The ensemble average of these 50 quantities (i.e., the time averages) is the observed expected retrieval cost and has a value of 2.468560. The theoretical expected retrieval cost in this case is 2.462485. The agreement between these values is remarkable.

6.5 Summary

In this chapter, we have studied the problem of reorganizing the elements of a linear list according to the MTF heuristic. Most of the current literature about the MTF rule (in fact all the data structuring heuristics) only considers the case in which the user's query distribution is stationary, or time invariant.

Two models of nonstationary query distribution have been presented in this chapter. The first is called the MSQG model for query generation — and is based on modeling the query generator *itself* as a Markov chain. In the second model, termed the RQG, the probability distribution used by the query generators at each

time instant is itself a *random vector*. The MTF rule has been analyzed under both these models. The results generalize the corresponding known results for stationary environments and coincide with the intuitive generalization. The theoretical results derived in this chapter have been experimentally verified.

Chapter 7

Multiple Access Queries

7.1 Introduction

In the previous chapter, we have studied the problem of reorganizing the elements of a Singly Linked List (SLL) whose elements were accessed, based on a non-stationary distribution. In this chapter, we study the impact of permitting the queries to consist of accesses to more than one element of the linear list. For this chapter, we shall require that the distribution of these generalized queries be time-invariant.

The problem of dynamically reorganizing a linear list with a view to minimize the average access cost has been well studied. In all the studies done to date, the situation analyzed (and which has been discussed rather thoroughly earlier) can be described as follows. Let $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ be a set of N distinct elements from which a linear list L is constructed. The elements of $\mathcal{R}$ are independently accessed, with a time invariant access distribution governed by the access probability vector $\mathcal{S} = [s_1, s_2, \dots, s_N]$ with the constraint that $\sum_{i=1}^N s_i = 1$. If the access probability vector $\mathcal{S}$ is known *a priori*, the list can be trivially organized in the optimal order, which corresponds to arranging the elements in the descending order of the access probabilities $\{s_i\}$ [45, 81].

The fundamental assumption made in all the research conducted to date is that the queries have been composed of **single** elements of the list. In this chapter, we wish to relax this assumption and study the implications of a generalized query

system. Specifically, we shall allow queries made against the system to consist of accesses to *any subset* of the elements of $\mathcal{R}$. These generalized queries are termed Multiple Access Queries (MAQ) and the underlying query generating mechanism will be referred to as the Multiple Access Query Generator (MAQG).¹

As a motivation to studying MAQs, consider the following scenario where such queries are encountered. We focus on a distributed environment with multiple users, where each user may initiate an access to the elements of a shared data structure stored on a centrally located data storage facility (typically the file server). The data structure being maintained at the file server, could be any one of the standard forms of organizing data, including a SLL, the Doubly-Linked List, A Binary Search Tree etc., or any other structure suitable to the applications. As a consequence of the fact that queries can originate at various sites, it is natural to expect that at any given instant, any subset of the data structure may be requested. In such a case, the file server would accumulate all the requests that occur in a "time unit", search for these elements, and subsequently report the relevant information to the respective sites. The problem investigated in this chapter is one of determining the optimal way in which the data structure can be maintained when queries of this type are encountered. Before we embark on a formal definition and analysis of MAQs, it is appropriate to mention few other contexts in which such queries are encountered.

The scenario of processing MAQs is not just encountered in distributed computing. Indeed, every database system uses a front-end which facilitates the data retrieval process and this serves as an interface between the user and the physical data. For most queries the response that is received is a table (relation) which indeed represents the sets of elements that satisfy the query. Thus, if the question was one of organizing the data structure which contains the actual physical data, the "systems manager" would have to arrange this physical data in such a manner that the average access time is minimized — the averaging being considered over all the possible sets which can be emitted as the response.

---

¹Although the MAQG is defined and analyzed for the SLL — one can easily envision the case when the accesses are set oriented and the data structure is arbitrary.

Having introduced the concept of MAQs and briefly motivated their occurrences and applications, it is not inappropriate to consider the aspect of specifying the distribution of queries. A straightforward method to model the MAQG comes easily to mind. To fully describe the MAQG it is evident that, we can specify a vector akin to $\mathcal{S}$, whose components comprise of the probabilities of accessing each of the possible 2^N subsets of $\mathcal{R}$. More formally, if ξ denotes an MAQ, then the MAQG is fully defined by completely specifying the probability $Pr(\xi)$ for all $\xi \subseteq \mathcal{R}$, where $\sum_{\xi} Pr(\xi) = 1$. It is obvious that this model of query generation subsumes the previous model in which only single elements are accessed.

There is one potential difficulty with specifying the MAQG by resorting to enumerating the access probabilities for all possible subsets of $\mathcal{R}$. In order to specify the *total* query probabilities $Pr(\xi)$ one would need to specify a vector of size 2^N , where N is the number of elements in $\mathcal{R}$. Although this method of describing the MAQG is both complete and exact (i.e., it is not an approximation) this is obviously unrealistic in terms of both time and space utilization. Ideally, we would like to have a simpler (albeit approximated) characterization of the query generation process, based on fewer parameters. We now present our model called the P-model for MAQs.

7.1.1 The P-model for MAQs

The model we propose in this chapter, will be called the P-model (the Parametrized model for the MAQG), and is characterized by exactly N parameters forming the vector $\mathcal{P} = [p_1, p_2, \dots, p_N]$. In the P-model for the MAQG, we assume that the records comprising a single query are statistically independent. Moreover, the queries at different time instants are also statistically independent. Finally, in any query ξ , we assume that the probability of a specific record R_i being requested is simply p_i . Notice that unlike the components of the S-vector used in the traditional single-access query generating mechanisms, the components of $\mathcal{P}$ need not sum to unity.

Using the above model it is easy to see that any arbitrary query $\xi \subseteq \mathcal{R}$ can be

fully described in terms of a bit-vector $\mathcal{B} = [b_1, b_2, \dots, b_N]$, where *each* component $b_i \in \{0, 1\}$ is a binary random variable obeying a Bernoulli distribution. Thus each bit b_i assuming a value in $\{0, 1\}$ signifies whether the record element R_i is included or not included in the query ξ . Throughout this chapter we shall make use of the obvious convention that $b_i = 1$ implies that R_i has been included in the query. Since the queries are assumed to be generated in an independent manner, this assertion reflects back to imply that the elements of $\mathcal{B} = [b_1, b_2, \dots, b_N]$ are chosen independently based on the distribution $\mathcal{P}$, and hence the probability of any particular query ξ being processed can be written down as:

$$Pr(\xi) = \prod_{i=1}^n p_i^{b_i} (1 - p_i)^{1-b_i} \quad (7.1)$$

To clarify the P-model we consider an example for the case when $N = 3$, and when $\mathcal{R} = \{R_1, R_2, R_3\}$. Then the set of possible queries is:

$$\{\{\}, \{R_1\}, \{R_2\}, \{R_3\}, \{R_1, R_2\}, \{R_1, R_3\}, \{R_2, R_3\}, \{R_1, R_2, R_3\}\}.$$

Thus an application of (7.1) suggests that under the P-model the probability of query $\xi = \{R_1, R_3\}$ occurring is exactly $p_1(1 - p_2)p_3$. Similar expressions can be written down for other MAQs.

Having completely defined the P-model for MAQs, we now consider the aspect of adaptively reorganizing a structure, when set queries of the type described above are encountered.

7.1.2 Data Reorganization Under MAQs

To fully comprehend data reorganization in the presence of generalized queries permitted by MAQ, it is convenient to draw some parallels from the former (single element access) query generation model. Observe that in this case, for the case of the linear list, the optimal order corresponds to the descending order of the access probabilities $\{s_i\}$. In the absence of this information, the most obvious way to approach the problem is to estimate the unknown parameters. After processing a sufficiently large number of queries, these estimates of s_i 's will be very "close" to

the *true* values and thus, with a high probability, the list can be arranged in the optimal order.

Coming now to the situation of MAQs, the question of obtaining an optimal ordering is unclear even if we are allowed to estimate a finite number of parameters from the “infinite” stream of queries. Furthermore, it is not even clear which parameters will serve as sufficient statistics in the estimation strategy. Indeed this brings us to the first result of our chapter, for in Theorem 7.1 we establish that the optimal (static) list can be obtained by arranging the list in the descending order of the p_i 's.

Since methods based on estimating the unknown parameters are generally discouraged in the literature, we next examine the possibility of employing data reorganizing heuristics in the context of MAQs. Two well known memoryless (ergodic) heuristics which have been discussed in Chapters 1 and 6 are the Move-to-Front (MTF) and Transposition (TR) schemes. To recapitulate, under the MTF heuristic, the accessed element is moved to the front of the list and the TR heuristic transposes the accessed element with the record that immediately precedes it. In both these heuristics, no reorganization is done if the accessed element happens to be at the front of the list. In this chapter, we shall propose two new heuristics for MAQs derived from the MTF and TR heuristics. These modified heuristics will be called MTF_TQS and TR_TQS respectively.

To describe the *modus operandus* of these schemes we begin by examining the query processing phase. This study naturally leads to an understanding of what constitutes the optimal order for the linear list. In this light it is shown that the expected access cost under the MTF_TQS heuristic is no more than $\pi/2$ times the optimal cost. This is a natural generalization of the corresponding result under the simpler model of query generation. Additionally, we show that MTF_TQS and TR_TQS are superior to the heuristics which merely employ the classical MTF and TR heuristics in conjunction with the stream consisting of requested records, and ignore the fact that the queries occur in sets. Thus we claim that there is considerable latent information in the set boundaries, and in this perspective, our

results represent a significant improvement from the traditional schemes.

We shall now proceed to consider various possible responses to queries generated by the MAQG.

7.2 Processing MAQs

Once a query ξ is presented, the system must search for the elements in ξ and report them to the user. We propose that this be done by choosing an element from the query at random and performing a linear search on the list starting from the front. The access cost of the element is exactly equal to the position of the element in the list, and it is also equal to the number of comparisons done (between the value of the query element and that of the list element) in order to find the element. Thus it will take one time unit to find an element which is at the head of a list, as opposed to N time units for the element which is at the tail of the list. Once the element is found, the next (as yet unsearched for) element in ξ is chosen randomly from the query and the search begins anew from the head of the list. The access cost associated with the query ξ is thus the sum of the access costs of all the elements in ξ .

An alternate strategy for processing could involve scanning the list from head to tail and comparing each element in the list with elements in the query until all the elements in ξ are located. The access cost associated with such a processing mechanism would now be quite different from the type of processing described above. Indeed, this cost would depend on what operations are permitted on the *query itself* (i.e. if we are permitted to sort the query etc.), and since this is a far more general problem we shall not discuss this aspect in the current work. The problem of reorganizing both the list and the query currently remains open.

Under the former model of query processing and the previously described model (the P-model) of query generation we shall now derive a fundamental result involving the optimal ordering of a linear list.

Theorem 7.1 *Let $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ be a set of elements, and $\mathcal{P}$ be the parameter for the P -model of the MAQG. Then if the query elements are processed sequentially, the optimal static list is obtained by maintaining the elements of $\mathcal{R}$ in the descending order of the components of $\mathcal{P}$.*

Proof:

In terms of notation let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also let $\Pi(i)$ be the element in the i^{th} position in the list, and $\Pi^{-1}(i)$ represent the position of element R_i in Π . The expected cost of accessing this list can be computed as the expected value of the cost given that the query is ξ . Thus,

$$E[\text{Cost}|\Pi] = \sum_{\xi} E[\text{Cost}|\xi, \Pi] Pr(\xi|\Pi). \quad (7.2)$$

Rewriting the expected cost of a query as the sum of costs of individual accesses, and using the fact that the probability of generating a particular query is independent of the list ordering we obtain:

$$E[\text{Cost}|\Pi] = \sum_{\xi} \sum_{i=1}^N \Pi^{-1}(i) b_i Pr(\xi), \quad (7.3)$$

where $Pr(\xi)$ is defined in equation (7.1) and b_i is a bit representing the event that $R_i \in \xi$. Notice that in (7.3), the summations on the right hand side are both finite. Hence we can interchange their order and obtain:

$$E[\text{Cost}|\Pi] = \sum_{i=1}^N \Pi^{-1}(i) \sum_{\xi} b_i Pr(\xi). \quad (7.4)$$

We note that b_i is a binary random variable and hence,

$$\sum_{\xi} b_i Pr(\xi) = E[b_i] = Pr(b_i = 1) = p_i,$$

and therefore (7.4) simplifies to:

$$E[\text{Cost}|\Pi] = \sum_{i=1}^N \Pi^{-1}(i) p_i. \quad (7.5)$$

One can see that given a specific list ordering Π , this expected cost is similar to the one obtained in the traditional model if the S -vector was the vector of access probabilities (i.e., where the s_i 's summed to one). Following Knuth's proof in [45], we can deduce that the optimal order corresponds to one in which the elements are arranged in the descending order of probabilities as given by $\mathcal{P}$. $\square$

Corollary 7.1 *For an MAQG whose access probability vector is $\mathcal{P}$ the expected cost for the optimal list, denoted by the ordering Π_{opt} , is given by:*

$$C_{opt}(\mathcal{P}) = E[Cost|\Pi_{opt}] = \sum_{i=1}^N ip_i. \quad (7.6)$$

As a result of Theorem 7.1, we know that if the entire $\mathcal{P}$ vector is known *a priori* we can trivially compute the optimal list order that minimizes the expected access cost. On the other hand, if the vector $\mathcal{P}$ is not known (which is the case in real life) we will have to resort to heuristics that aim to maintain the data structure in states for which the expected cost is close to the optimal expected cost. We shall now study ergodic heuristics for singly-linked-lists under the general query generation model. It is our aim to develop heuristics which reorganize the list without attempting to estimate the $\mathcal{P}$ vector. We propose natural extensions to the well known MTF and TR heuristics which satisfy the above conditions. Indeed, both of these require only the space to maintain the list itself. We shall begin by first examining the MTF heuristic.

7.3 Avenues for Studying MAQ Heuristics

In the preceding section, we have determined that the optimal ordering of records corresponds to the descending order of the components of the vector $\mathcal{P}$ characterizing MAQs. In this study, we are interested in investigating heuristics which do not estimate the vector $\mathcal{P}$ in order to minimize the expected access cost. Additionally we shall focus on memoryless ergodic heuristics [81], which generally reorganize the list, in response to each query.

Note that each of the queries emerging from MAQG consists of a *set* of elements. We shall refer to this stream of queries as the True Query Stream (TQS). At this point, two distinct directions of presenting the query information to the data reorganization heuristic exist. In one approach, the heuristic receives the queries as they occurred in the TQS together with the information about the boundaries between the sets. For example, if the first few queries in the TQS are $\{\{R_1\}, \{R_2\}, \{R_2, R_3\}, \{R_1, R_2, R_3\}\}$, the queries seen by the heuristic are $\{R_1\}$, $\{R_2\}$, and so on.

In the second approach to studying heuristics, we reduce the amount of information made available to the data reorganizing heuristic. We merely present the list of individual records accessed, without the information about the occurrence of the set boundaries in the TQS. Thus, for example in the TQS presented above, the sequence of records seen by the heuristic could be $\{R_1, R_2, R_2, R_3, R_1, R_2, R_3\}$. Notice that the stream of records accessed is still available except that the set boundaries have been “erased”. There is still one problem remaining to be resolved — namely, that of selecting the ordering of elements as presented in a *single set* to the heuristic. Since we do not have any *a priori* information about the record accesses, we should not discriminate between the records based on their “indices”. A fair strategy must ensure that all permutations of a given set query are equally likely. This stream consisting of accesses to individual records will be called the Serialized Query Stream (SQS).

In studying the behavior of heuristics dealing with single record accesses such as those observable in the SQS, we must determine two fundamental quantities. First of all we must determine the (asymptotic) probability of accessing the individual records of $\mathcal{R}$. Additionally, it is imperative that the statistical dependence (if any) between the successive record accesses be characterized. This information is definitely required in the analysis of *any* heuristic dealing with SQS. In order to obtain this we now proceed to study these aspects of the SQS, and in doing so we shall not pay any attention to the heuristic which may be employed by the data reorganization strategy.

7.3.1 Analysis of the Serialized Query Stream

In the introductory section, we had introduced the idea of query generation under the P-model. It was shown that any query $\xi \subseteq \mathcal{R}$ can be specified alternately in terms of a bit vector $\mathcal{B} = [b_1, b_2, \dots, b_N]$, where the bit b_i taking on the value 1 (0) indicates the presence (absence) of record R_i in the set under consideration. Given this bit-vector representation of the query ξ , the probability of generating this query is given by:

$$Pr(\xi) = \prod_{i=1}^n p_i^{b_i} (1 - p_i)^{1-b_i}$$

Note that in this model of query generation, the possibility that none of the records of $\mathcal{R}$ are accessed in a particular set, exists. In fact the probability of the occurrence of this event can be written down as $\prod_{i=1}^N (1 - p_i)$. The complementary event, namely that a non-null subset has been requested, occurs with a probability represented by the quantity η which is given by:

$$\eta = 1 - \prod_{i=1}^N (1 - p_i).$$

We now turn our attention to a simple characteristic of SQS. We first determine the probability that a "set boundary" occurs at a randomly chosen point in SQS. After evaluating this quantity, we shall use its form in the determination of the asymptotic probability of obtaining the various records of $\mathcal{R}$ in the SQS.

Let z_k , for $k = 1, 2, \dots, N$, denote the probability of encountering a set of cardinality k in the TQS. We can express z_k as follows:

$$z_k = \sum \prod_{j=1}^k p_{i_j} \left\{ \prod_{j=k+1}^N (1 - p_{i_j}) \right\}, \quad (7.7)$$

where $\langle i_1, i_2, \dots, i_N \rangle$ is a permutation of the integers in the set $\{1, 2, \dots, N\}$. The outermost summation is to be understood to be spanning over the $\binom{N}{k}$ possible choices of subsets of size k . As an illustration of (7.7), we present an example for the case when $N = 4$. For this case, z_2 has the form:

$$\begin{aligned} z_2 = & p_1 p_2 (1 - p_3)(1 - p_4) + p_1 p_3 (1 - p_2)(1 - p_4) + p_1 p_4 (1 - p_2)(1 - p_3) \\ & + p_2 p_3 (1 - p_1)(1 - p_4) + p_2 p_4 (1 - p_1)(1 - p_3) + p_3 p_4 (1 - p_1)(1 - p_2). \end{aligned}$$

Reverting to the general case, we observe that $\sum_{i=1}^N z_i = \eta$. Also we note that the occurrence of the null set is not observable in the SQS. In fact, as far as SQS is concerned, since the null query sets are neither observed nor processed, we can treat the events as occurring in the *conditional space* in which *only non-null query sets* are generated. We now define the quantity y_i as the *conditional* probability that a set with size i is generated, given that a non-null set has been generated in the TQS. By the laws of conditional probability,

$$y_i = \frac{z_i}{\eta}. \quad (7.8)$$

Note that $\sum_{i=1}^N y_i = 1$.

Our first concern now is to determine the probability of a set boundary having occurred at a random point in the SQS. To motivate the development of the mathematical model used to evaluate this quantity, let us suppose that the current point in time corresponds to the occurrence of a “set boundary”. If a set with k elements is now generated, the next set boundary occurs only k time units later. With some insight, this general philosophy can be embedded in a Markovian setting. Indeed, we endeavor to evaluate the probability of a set boundary occurring, using the markov chain $\mathcal{M}$ described as follows:

$$\mathcal{M} = (\Phi, M)$$

where,

Φ is the set of states $\{\phi_i | 0 \leq i \leq N - 1\}$. The state ϕ_i represents the event that i additional records have to be seen before the next set boundary is encountered. ϕ_0 is the state corresponding to the situation in which a set boundary has just been encountered. ϕ_1 is the event where the last record in a set has been processed and thus a set boundary will be encountered after one unit of time.

$M : \Phi \times \Phi \mapsto [0, 1]$ is the matrix of transition probabilities from ϕ_i to ϕ_j , for $0 \leq i, j \leq N - 1$.

Figure 7.1: The Markov chain capturing the information about set boundaries in SQS.

The graphical representation of the Markov chain is shown in Figure 7.1. Note that the matrix M is sparse, and its non-null components can be written down as:

$$\begin{aligned} M_{0,i} &= Pr(\phi_i|\phi_0) = y_{i+1}, \quad i = 1, 2, \dots, N-1 \\ M_{i,i-1} &= Pr(\phi_{i-1}|\phi_i) = 1, \quad i = 1, 2, \dots, N-1 \end{aligned}$$

Using the above representation, we shall now state and prove our first result which involves the computation of the probability of encountering a set boundary in the SQS.

Theorem 7.2 *The asymptotic probability that a set boundary occurs at a random instant in the SQS is given by:*

$$\pi_0^* = \frac{\eta}{\sum_{i=1}^N p_i}. \quad (7.9)$$

Proof:

The Markov Chain $\mathcal{M}$ is homogeneous, aperiodic, and recurrent, and is therefore ergodic [40]. As a result, the asymptotic distribution exists and is unique. We denote the asymptotic probability vector as $\pi^* = [\pi_0^* \pi_1^* \dots \pi_{N-1}^*]^T$, where π_i^* is the

asymptotic (i.e. stationary, limiting, or equilibrium) probability of the chain being in state ϕ_i .

The stationary distribution of this Markov chain can be found as the solution of the set of linear equations given by $M^T \pi^* = \pi^*$. Of course, these equations need to be augmented with the obvious constraint that the components of π^* must sum to unity. Using the sparseness of M , the components of π^* can be shown to obey the following scalar equations:

$$\begin{aligned} y_1 \pi_0^* + \pi_1^* &= \pi_0^* \\ y_{j+1} \pi_0^* + \pi_{j+1}^* &= \pi_j^* \quad 2 \leq j \leq N-2 \\ y_N \pi_0^* &= \pi_{N-1}^* \end{aligned} \tag{7.10}$$

The set of equations in (7.10) can be rewritten to yield all the components of π^* in terms of the common quantity π_0^* . Thus,

$$\begin{aligned} \pi_1^* &= (1 - y_1) \pi_0^* \\ \pi_{N-1}^* &= y_N \pi_0^* \\ \pi_{N-2}^* &= y_{N-1} \pi_0^* + \pi_{N-1}^* \\ &= (y_{N-1} + y_N) \pi_0^* \\ &\vdots \\ \pi_2^* &= (y_N + y_{N-1} + \cdots + y_3) \pi_0^* \end{aligned} \tag{7.11}$$

Combining the constraint $\sum_{i=0}^{N-1} \pi_i^* = 1$, with the set of equations in (7.11), we get:

$$\{1 + (1 - y_1) + (N - 2)y_N + (N - 3)y_{N-1} + \cdots + y_3\} \pi_0^* = 1 \tag{7.12}$$

Since $\sum_{i=1}^N y_i = 1$, we can equivalently modify (7.12) to be:

$$\left\{ \sum_{i=1}^N y_i + \sum_{i=2}^N y_i + \sum_{i=3}^N (i-2)y_i \right\} \pi_0^* = 1,$$

which simplifies to,

$$\left\{ \sum_{i=1}^N i y_i \right\} \pi_0^* = 1,$$

whence we obtain,

$$\pi_0^* = \frac{1}{\sum_{i=1}^N iy_i} = \frac{\eta}{\sum_{i=0}^N iz_i}. \quad (7.13)$$

Consider the denominator of (7.13). Observe that it consists of a single term, $\sum_{i=0}^N iz_i$, which is merely the expected number of elements in a random set query. Considering that the elements of $\mathcal{R}$ are independently chosen, the number of elements in a query, U , can be written down as:

$$U = \sum_{i=1}^N U_i,$$

where $U_i = 1(0)$ indicates the presense (absence) of record R_i in a query. Note that U_i 's are indicator random variables. Therefore, the expected number of elements in a query is given by $E[U] = \sum_{i=1}^N E[U_i] = \sum_{i=1}^N p_i$. Combining this result with (7.13) yields the required result in a straightforward fashion. $\square$

To aid in the presentation of our next major result, we introduce a key result.

Lemma 7.1 *Let $\gamma_k(\xi)$ denote the conditional probability that a superset of ξ of cardinality k , where $k \geq |\xi|$, is generated in the TQS, when the event is conditioned on the occurrence of the non-null set. Then, the following results hold:*

$$1. \quad \gamma_k(\xi) = \frac{1}{\eta} \cdot \sum_{\zeta \supseteq \xi, |\zeta|=k} \left[\prod_{R_i \in \zeta} p_i \right] \left[\prod_{R_i \in \mathcal{R} - \zeta} (1 - p_i) \right] \quad (7.14)$$

$$2. \quad \sum_{i=1}^N \gamma_l(\{R_i\}) = l \cdot \gamma_l(\{\}). \quad (7.15)$$

$$3. \quad \sum_{j \neq i} \gamma_l(\{R_i, R_j\}) = (l-1) \gamma_l(\{R_i\}). \quad (7.16)$$

Proof:

We note in passing that if $k < |\xi|$, the probability $\gamma_k(\xi)$ will be 0.

The probability $\gamma_k(\xi)$ can be written down by enumerating the various ways of forming the various supersets ζ of ξ , such that $|\zeta| = k$. The result given in (7.14) is obvious since the terms within the first and second square brackets represent the joint probability corresponding to the elements which have been included in the set

ζ , and the remaining elements of $\mathcal{R}$, i.e. $\mathcal{R} - \zeta$, respectively. This proves the first result of this Lemma.

Note that this definition for $\gamma_k(\xi)$ generalizes the definition for y_k originally defined in (7.8). In fact, y_k can be easily seen to be equivalent to $\gamma_k(\{\})$.

We now proceed to prove the second property stated in (7.15) above. Notice that $\gamma_l(\{R_i\})$ consists of $\binom{N-1}{l-1}$ terms, each term representing the contribution from one set of size l , which includes the element R_i . Moreover, the contribution of each such set (say κ) appears l times in the summation $\sum_{i=1}^N \gamma_l(\{R_i\})$. This is due to the fact that those indices i , which correspond to the records of κ contribute to the identical term. The total number of terms in the summation is $N \cdot \binom{N-1}{l-1}$, which is equivalent to $l \cdot \binom{N}{l}$. If we consider the grouping of terms as explained above, the result follows.²

The proof of the third property proceeds along the same lines as the previous proof. Once again, the number of terms in the term $\gamma_l(\{R_i, R_j\})$ is $\binom{N-2}{l-2}$. Therefore the total number of terms in the summation $\sum_{j \neq i} \gamma_l(\{R_i, R_j\})$ is $(N-1) \cdot \binom{N-2}{l-2}$, or $(l-1) \cdot \binom{N-1}{l-1}$. Let κ be a set which contributes to the quantity $\gamma_l(\{R_i, R_j\})$. The contribution from the same set κ appears $l-1$ times in the summation $\sum_{j \neq i} \gamma_l(\{R_i, R_j\})$. Grouping the terms as indicated above leads to (7.16).

□

Example 1.

It is not inappropriate to study an example which can help clarify the notation, philosophy and the proof of the second property stated in Lemma 7.1. Consider the situation in which $N = 4$, and $l = 2$. The $\mathcal{P}$ vector in this case has the value $[p_1, p_2, p_3, p_4]^T$. For convenience, we shall define the following terms, which represent the contributions from each of the $\binom{N}{l}$, or $\binom{4}{2} = 6$ sets.

$$t_1 = \frac{1}{\eta} p_1 p_2 (1 - p_3)(1 - p_4)$$

²An example which will help clarify the essential steps involved in the proof will be presented momentarily. We would rather maintain the formalism in the proof so as to retain the continuity of reasoning.

$$\begin{aligned}
t_2 &= \frac{1}{\eta} p_1 p_3 (1 - p_2)(1 - p_4) \\
t_3 &= \frac{1}{\eta} p_1 p_4 (1 - p_2)(1 - p_3) \\
t_4 &= \frac{1}{\eta} p_2 p_3 (1 - p_1)(1 - p_4) \\
t_5 &= \frac{1}{\eta} p_2 p_4 (1 - p_1)(1 - p_3) \\
t_6 &= \frac{1}{\eta} p_3 p_4 (1 - p_1)(1 - p_2).
\end{aligned} \tag{7.17}$$

Note that $\gamma_2(\{\})$ is by definition $\sum_{i=1}^6 t_i$.

Having defined these terms, we proceed to write the terms corresponding to $\gamma_2(\{R_i\})$ for $i = 1, 2, 3, 4$. These quantities can be written down in terms of the expressions t_1 through t_6 defined in (7.17) and have the form given below:

$$\begin{aligned}
\gamma_2(\{R_1\}) &= t_1 + t_2 + t_3 \\
\gamma_2(\{R_2\}) &= t_1 + t_4 + t_5 \\
\gamma_2(\{R_3\}) &= t_2 + t_4 + t_6 \\
\gamma_2(\{R_4\}) &= t_3 + t_5 + t_6.
\end{aligned} \tag{7.18}$$

Observe from (7.18) that the terms t_i , for $1 \leq i \leq 6$, appear in exactly two quantities of the form $\gamma_2(\{R_i\})$. In this example, the sum of the quantities on the LHS of (7.18) is $2 \sum_{i=1}^6 t_i$, which is the quantity $2\gamma_2(\{\})$. The generalization for other values of l should now be obvious to the reader.

By constructing a similar example, the reader can get a feeling for the correctness of the third property in Lemma 7.1. $\square$.

We now make use the conditional probabilities γ_k 's defined earlier, and derive a few more probabilities of interest. We now present our next result.

Lemma 7.2 *Let A_i denote the probability that the record R_i appears in the first position of a newly generated set query in SQS. Then the following results hold:*

$$1. \quad A_i = \sum_{l=1}^N \frac{\gamma_l(\{R_i\})}{l} \tag{7.19}$$

$$2. \quad \sum_{i=1}^N A_i = 1. \tag{7.20}$$

Proof:

The proof of the first result simply consists of enumerating all conditions under which the record R_i appears in the first position of a set. We note that the possible cardinalities for a query set are $1, 2, \dots, N$. The probability of generating a set of size l , which includes the element R_i is given by $\gamma_l(\{R_i\})$. Also, given that R_i occurs in a set of length l , the probability that it occurs in the first position is simply $1/l$ since all $l!$ permutations of a set of cardinality l are equally likely. Hence the probability that R_i appears first, in a set of size l , is simply given by $\gamma_l(\{R_i\})/l$. Summing this quantity for all possible values of l in the interval $[1, N]$, yields the expression stated in (7.19).

The proof of the second part is intuitively obvious since *one* of the records of $\mathcal{R}$ must come in the first position of a newly generated query in the SQS. An alternate way to conclude this is as follows. The LHS of (7.20) can be written as:

$$\begin{aligned}
 \sum_{i=1}^N A_i &= \sum_{i=1}^N \sum_{l=1}^N \frac{\gamma_l(\{R_i\})}{l} \quad (\text{from (7.19)}) \\
 &= \sum_{l=1}^N \frac{1}{l} \sum_{i=1}^N \gamma_l(\{R_i\}) \quad (\text{interchanging the order of summations}) \\
 &= \sum_{l=1}^N \gamma_l(\{\}) \quad (\text{by (7.15)}) \\
 &= \sum_{l=1}^N y_l \\
 &= 1.
 \end{aligned}$$

The lemma is now completely proved. $\square$

We state our next result which yields the asymptotic probability of observing the individual records of $\mathcal{R}$ in the SQS.

Theorem 7.3 *In the SQS, the asymptotic total probability of observing a record R_i is given by:*

$$p_i^* = \frac{p_i}{\sum_{j=1}^N p_j}$$

Proof:

Our aim is to compute the asymptotic probability³ of observing the record R_i in the SQS. We shall compute this quantity as the limiting value of the observing R_i at an arbitrary time instant. Indeed, it remains to be proved that the limit exists. With this information, the asymptotic can be easily computed. In terms of notation, let:

$\alpha_i(n)$ denote the event “record R_i is observed at time n ”.

$B(n)$ denote the event that a set boundary occurs at the time instant n in the SQS.

Also $\overline{B(n)}$ denotes the complement of the event $B(n)$.

$\pi_0(n)$ denote the probability that the Markov chain, $\mathcal{M}$ is in the state ϕ_0 (i.e. at a state boundary) at the time instant n . Observe that $\pi_0(n)$ is well defined because of the Markov Chain, $\mathcal{M}$. Moreover, its limiting value was shown to exist in Theorem 7.2, and has the form given by (7.9).

Using the above notation, the probability of the event $\alpha_i(n+1)$ can be written down as:

$$\begin{aligned} Pr[\alpha_i(n+1)] &= \sum_{j=1}^N Pr[\alpha_i(n+1) \cap \alpha_j(n)] \\ &= Pr[\alpha_i(n+1) \cap \alpha_i(n)] + \sum_{j \neq i} Pr[\alpha_i(n+1) \cap \alpha_j(n)] \end{aligned} \quad (7.21)$$

Consider the first term of (7.21), namely the probability $Pr[\alpha_i(n+1) \cap \alpha_i(n)]$. In terms of the notation already introduced, this quantity can be expressed as:

$$\begin{aligned} &Pr[\alpha_i(n+1) \cap \alpha_i(n)] \\ &= Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap B(n)] + Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap \overline{B(n)}], \end{aligned}$$

and since a single set in the TQS cannot include repeated elements, the second term on the RHS has a value of zero. Thus,

$$Pr[\alpha_i(n+1) \cap \alpha_i(n)]$$

---

³In essence the result states that the asymptotic (total) probability vector of accesses to the elements of $\mathcal{R}$ is simply the *normalized* form of the vector $\mathcal{P}$. Moreover, contrary to intuition, this result holds **only** asymptotically and it converges in a Markovian fashion.

$$\begin{aligned}
&= \Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap B(n)] \\
&= \Pr[\alpha_i(n+1) | \alpha_i(n) \cap B(n)] \cdot \Pr[\alpha_i(n) \cap B(n)] \quad (7.22)
\end{aligned}$$

The first term in the RHS of (7.22) is simply the probability that record R_i occurs in the first position of a newly observed set, and is the quantity A_i derived in (7.19) as per Lemma 7.2. We now focus on the second term of (7.22). First of all observe that if $n < N$ (i.e. the chain has just started) the possible instants for a query set initiation are $1, 2, \dots, n$. On the other hand, if $n > N$, the set boundary could have occurred at most N time units prior to n . Thus, in all cases, $\min(n, N)$ captures the number of time units in the past, where a new set could have begun.

$$\Pr[\alpha_i(n) \cap B(n)] = \sum_{k=1}^{\min(n, N)} \pi_0(n-k) \Pr \left[\begin{array}{l} R_i \text{ is the last element of set} \\ \text{of size } k \end{array} \right].$$

The probability of the event that R_i is the last element of a set of size k is quite simply $\gamma_k(\{R_i\})/k$. Therefore, we can write down:

$$\Pr[\alpha_i(n) \cap B(n)] = \sum_{k=1}^{\min(n, N)} \pi_0(n-k) \frac{\gamma_k(\{R_i\})}{k}. \quad (7.23)$$

As a result of Theorem 7.2 we know that $\lim_{n \rightarrow \infty} \pi_0(n)$ is π_0^* . Hence, observing that the term $\frac{\gamma_k(\{R_i\})}{k}$ is independent of n , and taking the limits of (7.23) as $n \rightarrow \infty$ we obtain,

$$\begin{aligned}
\lim_{n \rightarrow \infty} \{ \Pr[\alpha_i(n) \cap B(n)] \} &= \pi_0^* \sum_{k=1}^N \frac{\gamma_k(\{R_i\})}{k} \\
&= \pi_0^* A_i.
\end{aligned}$$

Incorporating the above result in (7.22), we get:

$$\lim_{n \rightarrow \infty} \{ \Pr[\alpha_i(n+1) \cap \alpha_i(n)] \} = \pi_0^* A_i A_i. \quad (7.24)$$

We now turn our attention to the evaluation of the second term of (7.21) and write:

$$\begin{aligned}
&\Pr[\alpha_i(n+1) \cap \alpha_j(n)] \\
&= \Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap B(n)] + \Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}] \quad (7.25)
\end{aligned}$$

By using arguments similar to those used to derive (7.24), the first term on the RHS of (7.25) can be shown to simplify to the following expression:

$$\lim_{n \rightarrow \infty} \{Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap B(n)]\} = \pi_0^* A_i A_j. \quad (7.26)$$

In the evaluation of the second term of (7.25), we know that a set boundary does not occur at time instant n . Therefore, the possible starting time instants for a set must belong to the interval $[n - \{\min(n, N) - 1\}, n - 1]$. Also note that when the set commences at the time instant $n - k$, the cardinality of the set is has to be in the closed interval $[k + 1, N]$. The probability of a set of size l (containing the records R_i and R_j) is given by $\gamma_l(\{R_i, R_j\})$. Moreover, utilizing the information that the distribution of permutations is uniform, the probability that records R_i and R_j are adjacent in the generated permutation of a set of cardinality l is $(l - 2)!/l!$ or $1/\{l(l - 1)\}$. Therefore we obtain:

$$Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}] = \sum_{k=1}^{\min(n, N)-1} \pi_0(n-k) \sum_{l=k+1}^N \frac{\gamma_l(\{R_i, R_j\})}{l(l-1)} \quad (7.27)$$

Taking the limits of both sides of (7.27) as $n \rightarrow \infty$ we obtain,

$$\begin{aligned} & \lim_{n \rightarrow \infty} \{Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}]\} \\ &= \sum_{k=1}^{N-1} \pi_0^* \sum_{l=k+1}^N \frac{\gamma_l(\{R_i, R_j\})}{l(l-1)} \\ &= \pi_0^* \sum_{l=2}^N \frac{\gamma_l(\{R_i, R_j\})}{l} \end{aligned} \quad (7.28)$$

Combining (7.28), (7.26), and (7.25) we obtain:

$$\lim_{n \rightarrow \infty} Pr[\alpha_i(n+1) \cap \alpha_j(n)] = \pi_0^* A_i A_j + \pi_0^* \sum_{l=2}^N \frac{\gamma_l(\{R_i, R_j\})}{l}.$$

Thus, the asymptotic probability of observing the record R_i can be written as:

$$\begin{aligned} p_i^* &= \lim_{n \rightarrow \infty} Pr[\alpha_i(n+1)] \\ &= \pi_0^* A_i A_i + \pi_0^* \sum_{j \neq i} A_i A_j + \sum_{j \neq i} \pi_0^* \sum_{l=2}^N \frac{\gamma_l(\{R_i, R_j\})}{l} \\ &= \pi_0^* A_i \sum_{j=1}^N A_j + \pi_0^* \cdot \sum_{l=2}^N \frac{1}{l} \sum_{j \neq i} \gamma_l(\{R_i, R_j\}) \end{aligned} \quad (7.29)$$

Using (7.20), (7.16), and (7.29), we see that:

$$\begin{aligned}
 p_i^* &= \pi_0^* A_i + \pi_0^* \cdot \sum_{l=2}^N \frac{(l-1)\gamma_l(\{R_i\})}{l} \\
 &= \pi_0^* \left\{ \sum_{l=1}^N \frac{\gamma_l(\{R_i\})}{l} + \sum_{l=2}^N \frac{(l-1)\gamma_l(\{R_i\})}{l} \right\} \\
 &= \pi_0^* \left\{ \sum_{l=1}^N \gamma_l(\{R_i\}) \right\} \\
 &= \pi_0^* \frac{p_i}{\eta}
 \end{aligned}$$

But in Theorem 7.2, it was shown that $\pi_0^* = \eta / (\sum_i p_i)$. Hence, the expression for p_i^* can be further simplified to the form:

$$p_i^* = \frac{p_i}{\sum_{j=1}^N p_j},$$

and the theorem is proved. $\square$

Thus far we have completely characterized the SQS in terms of the asymptotic probabilities of accessing the individual records. Moreover, it is easy to see that successive record accesses are *statistically dependent*. Given this dependence between queries, the analysis of ergodic heuristics is far from simple. In fact the only results for dependent queries are due to Lam *et. al.* in their pioneering paper [48], and the results presented in Chapter 6. The general technique to model these systems is to study the composite chain, whose state space is the vector product of the $N!$ states of the linear list, and the N states of the query generator. The exact solution therefore involves the evaluation of $N \cdot N!$ stationary probabilities. This is impractical except for very small values of the parameter N , the number of records in the list. For the special case of the MTF heuristic, Lam *et. al.* were able to obtain closed form expressions for the expected cost, in terms of the mean times for first passage of the reversed time chain representing the query generator. The analysis of the TR heuristic remains open.

Hereafter, we shall refer to the classical MTF heuristic operating in conjunction with the SQS as MTF_SQS. In principle, we have the parameters needed to commence the analysis of MTF_SQS. Due to the complexity of the matrix capturing the

dependence between queries, it is not feasible to follow the approach taken by Lam *et. al.* and compute the expected cost. Rather than explicitly analyze MTF_SQS we shall present a variant of the classical MTF, which interacts with the TQS. This heuristic will be called MTF_TQS. We shall first analyze the behavior of MTF_TQS, and then compare it to the performance of MTF_SQS. In this special case, we can show without resorting to the techniques used by Lam *et. al.*, that MTF_TQS indeed yields a lower expected cost than MTF_SQS.

7.4 The Move-to-Front Heuristic for MAQs

We shall now describe the proposed MTF_TQS heuristic for the MAQG and subsequently proceed to analyze it. As is well known, the general characteristic of the MTF scheme is that it moves the accessed elements towards the front of the list. Since we are encountering the request for multiple elements in a single query it is natural to move all of them to the front of the list. Of course, if an element is already at the head of the list it need not be moved, and thus in such a case no list reorganization need be done. There is however one major note of difference between the current scheme and the traditional MTF. If we have a query accessing multiple elements of $\mathcal{R}$, there is no unique way to move them to the front of the list. In the absence of such a natural ordering our algorithm proposes that the sequence of moving the elements be chosen randomly in such a way that all permutations of a given query are equally likely. The whole procedure is algorithmically given in Algorithm MTF_TQS and is illustrated in Figure 7.2.

Algorithm MTF-TQS:**Input:**

A list consisting of elements in the set R and an infinite sequence $\{\xi\}$ of MAQs. The algorithm gets the query ξ from the query processor by invoking `GetNextQuery`. ξ is itself stored as a list.

Output:

The reorganized list according to the MTF-TQS scheme.

Assumptions:

It is assumed that the following elementary procedures are accessible to the algorithm:

- (i) `Head (aList)` - which returns the element at the head of the list.
- (ii) `DeleteHead (aList)` - which removes the head of *aList* and returns the tail.
- (iii) `PermuteList (aList)` - which uniformly permutes the given list.
- (iv) `Move-To-Front (aList, element)` - which moves the only match found for *element* in *aList* to the front.

Method:

```

 $\xi$  := GetNextQuery
PermuteList ( $\xi$ )
while not EmptyList ( $\xi$ ) do
    Move-To-Front (aList, Head ( $\xi$ ))
     $\xi$  := DeleteHead ( $\xi$ )
endwhile

```

End Algorithm MTF-TQS**Program 7.1: The MTF-TQS heuristic for MAQGs**

Observe that if the queries consist of only accesses to single elements of the list, this reorganization scheme is no different from the classical MTF. Given this qualitative description of the new MTF scheme we now proceed to examine the analytic properties of the scheme.

Figure 7.2: Diagram showing a sample seven element list before and after a query is processed. The reorganization is done using the MTF-TQS scheme. In this case the query is $\{2, 3\}$ and both possible outcomes are shown. In the first case 2 is moved first then 3, in the second case 3 is moved first, followed by 2.

Theorem 7.4 *For the MTF-TQS heuristic responding to a MAQG the asymptotic probability of an element R_i preceding another element R_j is given by ${}_iP_j^*$, where:*

$${}_iP_j^* = \frac{p_i - 0.5p_i p_j}{p_i + p_j - p_i p_j} \quad (7.30)$$

Proof:

We prove this result by deriving a recursive relation for the probability of a record R_i preceding the record R_j . Let ${}_iP_j(n)$ denote this probability at time instant n . We now calculate the conditional probability ${}_iP_j(n+1)$ given ${}_iP_j(n)$.

Let $\xi(n)$ be the query at time n . We have four mutually exclusive and collectively exhaustive cases. First of all, ${}_iP_j(n+1)$ is 0 if R_j was accessed and R_i was not accessed, since in that case R_j would be moved to the front while R_i would be unmoved. Thus, R_i will never precede R_j . In the second case, if R_i was accessed and R_j was not accessed the roles are interchanged and R_i will always precede R_j since it will be moved to the front at some point in the reorganization. The third case covers the possibility of neither R_j nor R_i being accessed in which case the probability of R_i preceding R_j is the same as it was before, since both elements R_i and R_j remain unmoved. Finally if both R_i and R_j are accessed, both of them will be moved to the front. However, since all permutations of $\xi(n)$ are equally likely to dictate the reorganization strategy, R_i will be moved after R_j with probability 0.5. Thus R_i will ultimately precede R_j with probability 0.5, and R_j will precede R_i with probability 0.5. Formalizing this we obtain:

$${}_iP_j(n+1) = \begin{cases} 0 & \text{if } R_j \in \xi(n), R_i \notin \xi(n) \\ 1 & \text{if } R_i \in \xi(n), R_j \notin \xi(n) \\ {}_iP_j(n) & \text{if } R_i, R_j \notin \xi(n) \\ 0.5 & \text{if } R_i, R_j \in \xi(n) \end{cases} \quad (7.31)$$

Using (7.1) we can calculate the probabilities with which each of these events can occur. Thus⁴,

---

⁴w.p. is an abbreviation for "with probability".

$${}_iP_j(n+1) = \begin{cases} 0 & \text{w.p. } (1-p_i)p_j \\ 1 & \text{w.p. } p_i(1-p_j) \\ {}_iP_j(n) & \text{w.p. } (1-p_i)(1-p_j) \\ 0.5 & \text{w.p. } p_ip_j \end{cases}$$

From the above the laws of probability yield that the quantity ${}_iP_j(n+1)$ given ${}_iP_j(n)$ has the form:

$$\begin{aligned} {}_iP_j(n+1) &= p_i(1-p_j) + {}_iP_j(n)(1-p_i)(1-p_j) + 0.5p_ip_j \\ &= (p_i - 0.5p_ip_j) + {}_iP_j(n)(1-p_i)(1-p_j) \end{aligned} \quad (7.32)$$

Thus ${}_iP_j(n)$ obeys a simple first order difference equation. Its general solution is given by:

$${}_iP_j(n) = \{(1-p_i)(1-p_j)\}^n {}_iP_j(0) + \frac{p_i - 0.5p_ip_j}{p_i + p_j - p_ip_j} \cdot [1 - \{(1-p_i)(1-p_j)\}^n] \quad (7.33)$$

It can thus be seen that the sequence $\{{}_iP_j(n)\}$ converges when $0 < (1-p_i)(1-p_j) < 1$. Its asymptotic value can be easily shown to be given by (7.30).

We note that the above condition for convergence is not satisfied only when $p_i = p_j = 0$ (or 1). When $p_i = p_j = 0$, ${}_iP_j(n)$ is equal to ${}_iP_j(0)$, for all values of n . In the case when the common value of p_i, p_j is 1, ${}_iP_j(n)$ is equal to 0.5 for all values of n . $\square$

We now derive a lower bound for the quantity ${}_iP_j^*$ derived in Theorem 7.4.

Lemma 7.3 *For all i, j , whenever $p_i > p_j$, a lower bound for ${}_iP_j^*$ is given by:*

$${}_iP_j^* > \frac{p_i}{p_i + p_j}$$

Proof:

A simple rearrangement of the expression for ${}_iP_j^*$ in (7.30) yields,

$$\begin{aligned} {}_iP_j^* &= \frac{1}{1 + \frac{p_j - 0.5p_ip_j}{p_i - 0.5p_ip_j}} \\ &> \frac{1}{1 + \frac{p_j}{p_i}} \quad \left(\text{Since } \frac{p_j - 0.5p_ip_j}{p_i - 0.5p_ip_j} < \frac{p_j}{p_i} \text{ whenever } p_i > p_j. \right) \\ &> \frac{p_i}{p_i + p_j}, \end{aligned}$$

and the result follows immediately. In passing we note that whenever $p_i = p_j$ the quantity ${}_jP_i^*$ equals 0.5. $\square$

Using the result of Lemma 7.3 we can now compare the performance of the MTF-TQS scheme with the behavior of the optimal static list.

Theorem 7.5 *The expected cost of the MTF-TQS scheme under the MAQG is at most $\pi/2$ times the optimal cost.*

Proof:

Let $C_{MTF-TQS}(\mathcal{P})$ denote the expected cost of accessing elements of the linear list under the MAQG when the MTF heuristic is being used. $C_{OPT}(\mathcal{P})$ denotes the similar quantity for the optimal static ordering. Hence we wish to prove that:

$$\frac{C_{MTF-TQS}(\mathcal{P})}{C_{OPT}(\mathcal{P})} \leq \frac{\pi}{2}.$$

We first compute the expected cost under the MTF heuristic. To simplify the computation we compute the conditional expected cost, given that the ordering is a specific permutation Π . This quantity has been derived earlier and is:

$$E[Cost|\Pi] = \sum_{i=1}^N \Pi^{-1}(i)p_i. \quad (7.34)$$

Taking expectations of (7.34) we obtain:

$$C_{MTF-TQS}(\mathcal{P}) = E[E[Cost|\Pi]] = \sum_{i=1}^N E[\Pi^{-1}(i)]p_i. \quad (7.35)$$

We note that $E[\Pi^{-1}(i)]$ is simply the expected position of record R_i in the list, when the MTF heuristic is being used. Analogous to the derivation of Hendricks [32], we now extend our arguments to yield:

$$E[\Pi^{-1}(i)] = 1 + \sum_{j \neq i} {}_jP_i^*,$$

and hence,

$$C_{MTF-TQS}(\mathcal{P}) = \sum_{i=1}^N p_i \left\{ 1 + \sum_{j \neq i} {}_jP_i^* \right\}. \quad (7.36)$$

Without any loss of generality let us assume that $p_i > p_j$ for all $i < j$. Also for all $i \neq j$, let ${}_iP_j$ denote quantities which obey the following conditions:

$$\begin{aligned} {}_iP_j + {}_jP_i &= 1 \\ {}_iP_j^* &\geq {}_iP_j \quad \forall i < j. \end{aligned}$$

Then following arguments similar to Hendricks [32] we can easily prove⁵ that:

$$C_{MTF-TQS}(\mathcal{P}) \leq \sum_{i=1}^N p_i \left\{ 1 + \sum_{j \neq i} {}_jP_i \right\}. \quad (7.37)$$

By Lemma 7.3 we have identified lower bounds for the quantity ${}_iP_j^*$ for $i < j$. Thus, we can set:

$${}_iP_j = \frac{p_i}{p_i + p_j} \quad \forall i < j,$$

and (7.37) becomes:

$$C_{MTF-TQS}(\mathcal{P}) \leq \sum_{i=1}^N p_i \left\{ 1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j} \right\}. \quad (7.38)$$

Notice that (7.38) is of a form similar to the cost of the classical MTF rule. (7.38) can be easily simplified to yield:

$$C_{MTF-TQS}(\mathcal{P}) \leq 2 \sum_{i \leq j} \frac{p_i p_j}{p_i + p_j}. \quad (7.39)$$

But the optimal cost for the SLL under MAQG was previously derived and is given by:

$$C_{OPT}(\mathcal{P}) = E[Cost|\Pi_{opt}] = \sum_{i=1}^N i p_i,$$

and thus we have:

$$\frac{C_{MTF-TQS}(\mathcal{P})}{C_{OPT}(\mathcal{P})} \leq \frac{2 \sum_{i \leq j} \frac{p_i p_j}{p_i + p_j}}{\sum_{i=1}^N i p_i}. \quad (7.40)$$

---

⁵The new set of quantities ${}_iP_j$ can be interpreted in Hendricks' proof as the set of probabilities of record R_i preceding R_j , according to some arbitrary heuristic.

Observe that Chung *et. al.* proved that the right hand side of (7.40) is always less than or equal to $\pi/2$ [13]. This proves the theorem. $\square$

So far we have analyzed the performance of MTF_TQS heuristic when operating in the presence of set queries. The reader will recall this scheme had access to the TQS. Moreover, the heuristic utilized the information about the boundary between the various set queries. The issue which we investigate next concerns whether knowledge of these boundaries indeed results in a lower expected cost for query accesses. In other words, we wish to compare the performance of the two heuristics MTF_TQS and MTF_SQS.

Before we embark on a comparison of MTF_TQS and MTF_SQS, it is useful to form a clearer idea of their operation. Observe that in MTF_TQS, whenever the list configuration is $L(n)$, all the elements of a specific query $\xi(n)$ are searched in this list. In other words, the list reorganization takes place only when the desired elements have been located. Moreover, in the list reorganization process, the list “visits” several intermediate states of the underlying Markov Chain. The key characteristic which differentiates these heuristics is that in the case of MTF_TQS, these “intermediate states” are not observable. In the case of MTF_SQS, the list $L(n)$ is transformed *immediately* upon locating an element from $\xi(n)$, and the search for the remaining elements of $\xi(n)$ is done in the *transformed* list. Note however that after processing all the elements of a set query both heuristics will produce the same final list, since they perform the same sequence of data reorganization operations.

We now state our next result which claims the superiority of MTF_TQS over MTF_SQS. Informally speaking, the result claims that there is some latent information in the set boundaries and if this information is utilized, superior performance is attainable.

Theorem 7.6 *For all access distributions, the heuristic MTF_TQS yields a lower expected access cost than the heuristic MTF_SQS.*

Proof:

Let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also, let $\text{Cost}_{\text{MTF\_TQS}}$

and $\text{Cost}_{\text{MTF\_SQS}}$ be the random variables denoting the access costs for queries, under the heuristics MTF-TQS and MTF-SQS respectively. The expected cost $C_{\text{MTF\_TQS}}$ can be computed as follows:

$$C_{\text{MTF\_TQS}} = E[E[\text{Cost}_{\text{MTF\_TQS}}|\xi, \Pi]]. \quad (7.41)$$

A similar procedure can be followed to obtain the expected cost $C_{\text{MTF\_SQS}}$ under the MTF-SQS heuristic.

We focus on the aspect of computing the conditional expected cost, given that the list is represented by Π . Using the notation that $\Pi(i)$ refers to the i^{th} element of Π , with no loss of generality, we can represent a general query set ξ as $\{\Pi(m_1), \Pi(m_2), \dots, \Pi(m_{|\xi|})\}$, where $1 \leq m_i \leq N$, and $m_i < m_{i+1}$. In other words, m_i 's are the indices of elements of Π which appear in ξ , and furthermore they represent the order in which they appear in Π .

Notice that since MTF-TQS uses only the list Π to locate the elements, $E[\text{Cost}_{\text{MTF\_TQS}}|\xi, \Pi]$ is clearly $\sum_{i=1}^{|\xi|} m_i$, where the expectation is done w.r.t. the uniform distribution of all $|\xi|!$ permutations of ξ . We now turn our attention to the behavior of MTF-SQS. The cost incurred by MTF-SQS is greatly influenced by the order in which the elements of ξ appear in the SQS. In other words, we need to consider the $|\xi|!$ possible permutations (or serializations) of ξ .

First of all, consider the simple case in which the generated serialization of ξ corresponds exactly to the order in which elements are present in Π . In this case, although MTF-SQS moves the accessed records to the front of the list, the positions of all the other records are not altered, since they appear later in the list than the recently accessed element. Thus, in this case the cost incurred happens to be $\sum_{i=1}^{|\xi|} m_i$. In all other cases, the random permutation of ξ does not correspond to the order in which the elements are found in Π . The effect of accessing elements in a sequence other than in the order in which they are in Π , results in *at least one* record moving to position greater than its initial position. Consequently, we claim that the total access cost will be strictly greater than $\sum_{i=1}^{|\xi|} m_i$. This immediately yields the result that:

$$E[\text{Cost}_{MTF\_SQS}|\xi, \Pi] > \sum_{i=1}^{|\xi|} m_i.$$

Combining this equation with the similar cost for the MTF_TQS heuristic, we obtain:

$$E[\text{Cost}_{MTF\_SQS}|\xi, \Pi] > E[\text{Cost}_{MTF\_TQS}|\xi, \Pi]. \quad (7.42)$$

Taking expectations of (7.42), by considering the distribution of queries, we deduce:

$$E[\text{Cost}_{MTF\_SQS}|\Pi] > E[\text{Cost}_{MTF\_TQS}|\Pi]. \quad (7.43)$$

Taking expectations a second time, and observing that (7.41) is true, we get the desired result that:

$$C_{MTF\_SQS} > C_{MTF\_TQS}.$$

and the theorem is proved. $\square$

7.5 The Transposition Heuristic for MAQs

In addition to the MTF_TQS heuristic already described we now present a variant of the classical TR heuristic where the accessed elements are transposed with the elements directly preceding them. This data reorganization scheme will be termed TR_TQS. As before, we encounter many elements in each query, and each of these elements has to be individually transposed. We define this heuristic, in a manner analogous to MTF_TQS and permute the *query* so that each element in the query has an equal probability of being transposed first, second, or last. Figure 7.3 depicts this scheme for a list of seven elements. Note that in certain cases the transposition scheme may leave the list unaffected, and this could occur whenever all elements of the query are adjacent in the list. Additionally if the element to be moved is already at the head of the list, the list is left untouched. The algorithm is formally given below.

Algorithm TR_TQS:**Input:**

A list consisting of elements in the set R and an infinite sequence $\{\xi\}$ of MAQs. The algorithm gets the query ξ from the query processor by invoking `GetNextQuery`. ξ is itself stored as a list.

Output:

The reorganized list according to the scheme TR_TQS.

Assumptions:

It is assumed that the following elementary procedures are accessible to the algorithm:

- (i) `Head (aList)` - which returns the element at the head of the list.
- (ii) `DeleteHead (aList)` - which removes the head of *aList* and returns the tail.
- (iii) `PermuteList (aList)` - which uniformly permutes the given list.
- (iv) `Transpose (aList, element)` - which transposes the only match found for *element* in *aList* with the element directly preceding element.

Method:

```
 $\xi := \text{GetNextQuery}$   
PermuteList ( $\xi$ )  
while not EmptyList ( $\xi$ ) do  
  Transpose (aList, Head ( $\xi$ ))  
   $\xi := \text{DeleteHead ( $\xi$ )}$   
endwhile
```

End Algorithm TR

Program 7.2: The TR_TQS heuristic for MAQGs

Figure 7.3: Diagram showing a sample seven element list before and after a query is processed. The reorganization is done using the TR.TQS scheme. In this case the query is $\{2, 3\}$ and both possible list configurations are shown. Note that the list will be unchanged if $\xi = \{2, 3\}$ and the transposition of 3 precedes the transposition of 2. If, on the other hand the query was $\{2, 6\}$, both possible permutations lead to the same final outcome.

In the traditional case the analysis for the TR heuristic is typically done by taking advantage of the time reversibility of the underlying Markov Chain. In fact, in the traditional case it can be shown that the chain is time reversible because the probabilities of going from one permutation to another in the forward direction exactly equal the probabilities of reversing the transition in the reverse chain.

This however, need not be the case when the TR.TQS heuristic operates under the MAQG. The reason for this is as follows. When a query ξ is presented to the scheme, if the query was processed in a particular order it would be equivalent to running the traditional TR heuristic for the sequence of queries specified by the order. Notice that in such a case one could have envisaged an argument to show that the probabilities associated with the reverse direction are identical if the query was processed in the reverse order. However, this argument could be fallacious because, whereas in the traditional TR scheme we are permitted to have consecutive accesses to the same element, in the current scheme it is assumed that the elements of ξ are all distinct. Thus it is not clear that the underlying Markov Chain is time reversible and consequently we believe that there is no straightforward technique to analyze the behavior of TR under an MAQG. Of course, the naive method is always possible which involves listing all the $N!$ possible states and subsequently computing all the $N!$ stationary probabilities.

While the explicit expected cost for the TR.TQS heuristic cannot be trivially written, we nevertheless can compare its expected cost to the classical heuristic dealing with the SQS. We shall refer to *this* variant of TR as TR.SQS. We now state our final result which claims the superiority of TR.TQS over TR.SQS in all random environments. Just as in the case of the two variants of the MTF heuristic, we once again observe that the information about set boundaries can be utilized to attain a lower expected access cost for queries.

Theorem 7.7 *For all access distributions, the heuristic TR.TQS yields a lower expected access cost than the heuristic TR.SQS.*

Proof:

As in Theorem 7.6, let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also,

let Cost_{TR_TQS} and Cost_{TR_SQS} be the random variables denoting the access costs for queries, under the heuristics TR.TQS and TR.SQS respectively. The expected cost C_{TR_TQS} can be computed as follows:

$$C_{TR\_TQS} = E[E[\text{Cost}_{TR\_TQS}|\xi, \Pi]]. \quad (7.44)$$

A similar procedure can be followed to obtain the expected cost $C_{\text{Cost}_{TR\_SQS}}$ under the TR.SQS heuristic.

We focus on the aspect of computing the conditional expected cost, given that the list is represented by Π . Recapitulating the semantics of the notation used in Theorem 7.6, $\Pi(i)$ refers to the i^{th} element of Π , with no loss of generality, we can represent a general query set ξ as $\{\Pi(m_1), \Pi(m_2), \dots, \Pi(m_{|\xi|})\}$, where $1 \leq m_i \leq N$, and $m_i < m_{i+1}$. In other words, m_i 's are the indices of elements of Π which appear in ξ , and furthermore they represent the order in which they appear in Π .

Just like the MTF.TQS heuristic, TR.TQS uses only the list Π to locate the elements, and consequently $E[\text{Cost}_{TR\_TQS}|\xi, \Pi]$ is clearly $\sum_{i=1}^{|\xi|} m_i$, where the expectation is done w.r.t. the uniform distribution of all $|\xi|!$ permutations of ξ . We now turn our attention to the behavior of TR.SQS. The cost incurred by TR.SQS is greatly influenced by the order in which the elements of ξ appear in the SQS. In other words, we need to consider the $|\xi|!$ possible permutations (or serializations) of ξ .

Observe that TR.SQS swaps the most recently accessed record with the element which immediately precedes it, and hence it does not disturb the positions of the records which appear after the accessed record. Moreover, accesses to any of the records do not cause TR.SQS to move any other record which is yet to be accessed, closer to the beginning. As a result, the cost incurred in processing a query can never be less than $\sum_{i=1}^{|\xi|} m_i$. With this information alone we can conclude that TR.TQS performs at least as well as TR.SQS. Our aim is to exclude the possibility of equality in average access cost, thereby establishing that TR.TQS is strictly better than TR.SQS.

First of all, consider the simple case in which the generated serialization of ξ corresponds exactly to the order in which elements are present in Π . In this case,

although TR_SQS moves the accessed records towards the front of the list, the positions of all the other records are not altered, since they appear later in the list than the recently accessed element. Thus, in this case the cost incurred is exactly $\sum_{i=1}^{|\xi|} m_i$. In all other cases, the random permutation of ξ does not correspond to the order in which the elements are found in Π , and our endeavor is now to demonstrate the existence of cases when the total access cost can be strictly greater than $\sum_{i=1}^{|\xi|} m_i$.

Consider a scenario in which the serialized permutation of ξ corresponds to the reverse of the order in which these elements occur in Π . The last two elements in this generated serialization are $\Pi(m_{|\xi|-1})$ and $\Pi(m_{|\xi|})$. If these two elements are not adjacent in Π , accessing $\Pi(m_{|\xi|})$ does not displace the other record and the total cost of accessing these records is not greater than the minimum possible cost. On the other hand, if these two records are adjacent, they will be swapped by TR_SQS resulting in the record $\Pi(m_{|\xi|-1})$ moving to the position $m_{|\xi|}$, thereby incurring a greater cost. Since we have shown that there exist serializations of ξ which incur a strictly greater cost than the lowest possible cost, the expected cost given a particular query and an ordering, will be strictly greater under TR_SQS. In other words,

$$E[\text{Cost}_{\text{TR\_SQS}}|\xi, \Pi] > \sum_{i=1}^{|\xi|} m_i.$$

Combining this equation with the similar cost for the TR_TQS heuristic, we obtain:

$$E[\text{Cost}_{\text{TR\_SQS}}|\xi, \Pi] > E[\text{Cost}_{\text{TR\_TQS}}|\xi, \Pi]. \quad (7.45)$$

The required result follows by taking the expectations of (7.45) two times and making use of (7.44). $\square$

We shall now proceed to present simulation results which demonstrate the power of the MTF_TQS and TR_TQS heuristics when responding to an MAQG.

7.6 Experimental Results

Experiments were conducted to verify the theoretical result presented in this chapter. The aim of the experiments was to compare the expected cost of queries under

the MTF and TR heuristics when compared to the cost of the optimal (static) list. In summary, the experiments consisted of generating a large number of queries, based on a chosen vector $\mathcal{P}$, and then averaging the cost incurred using the different heuristics. The details of the experiments conducted are as follows.

Since the MAQG generation strategy was based on the parameter $\mathcal{P}$, the first step in the simulation involved assigning the vector a numerical value. Since there is no loss of generality in assuming that the components of $\mathcal{P}$ are in a descending order, the vectors used in the simulations obeyed this relation. As opposed to generating random vectors for $\mathcal{P}$, we chose three different types of variations in the components of $\mathcal{P}$. These variations are listed below and are the generalized⁶ versions of the well-known distributions:

(i) **Generalized Linear:** The components of $\mathcal{P}$ are given by

$$p_i = c - k \cdot i \quad (7.46)$$

where $i = 1, 2, \dots, N$ and c, k are suitable positive constants satisfying $c - kN > 0$. The latter condition is imposed merely to ensure that the components of $\mathcal{P}$ are positive quantities.

(ii) **Generalized Zipf's:** The components of $\mathcal{P}$ are given by:

$$p_i = c + \frac{k}{i} \quad (7.47)$$

where c , and k are again suitable positive constants chosen so as to render each component of $\mathcal{P}$ to be a probability.

(iii) **Generalized Exponential:** In this case, the components of the $\mathcal{P}$ are given by:

$$p_i = c + \frac{k}{e^i} \quad (7.48)$$

---

⁶We call these generalized because the components of $\mathcal{P}$ obey the fundamental property of the law but need not sum to unity.

where e is the base of the natural logarithms.

After determining the value of N , and the type of variation among the components of $\mathcal{P}$ (i.e., one of Generalized Linear/Zipf/Exponential), the components p_1 and p_N were fixed to be 0.9 and 0.4 respectively. This choice immediately fixes the values of the constants c and k in the equations (7.46) through (7.48) and these values subsequently assist in the evaluation of $p_2 \dots p_{N-1}$.

As an example, consider the case when $N = 6$. The vectors obtained for the cases of Linear and Zipf's variation in the components of $\mathcal{P}$ are given below:

$$\mathcal{P}(\text{Linear}) = [0.9, 0.8, 0.6, 0.7, 0.6, 0.5, 0.4]^T$$

$$\mathcal{P}(\text{Zipf's}) = [0.9, 0.6, 0.5, 0.45, 0.42, 0.4]^T$$

Using the vectors chosen as described above, queries were generated. The initial configuration of the list was randomly chosen to be one of the $N!$ permutations. Subsequently the heuristic being evaluated (i.e. either MTF or TR) was invoked for an initial period of 10,000 queries before the access costs were monitored. This initial waiting period served to ensure that the Markov chains had converged. The reported results consist of the ensemble average (over 100 experiments) of the time average of 2,500 queries following the above mentioned "transient" phase.

Various experiments were conducted for values of N ranging from 4 to 12, in steps of 2. The results for the MTF.TQS and TR.TQS heuristics for the above values of N and three types of distributions specified earlier have been reported in Tables 7.1-7.6. Observe that that Tables 7.1-7.3 report the %-error between the *theoretical expected cost* for the MTF.TQS heuristic and the optimal cost.

Consider Table 7.1 which report the results obtained for the MTF.TQS heuristic when there was a linear gradation among the components of $\mathcal{P}$. The experimentally obtained average cost agrees well with the theoretically predicted value. Moreover, if we examine the case when $N = 6$, the expected cost under the MTF.TQS heuristic had a value of 13.0685, whereas the optimal cost was 11.9. This implies that the expected cost when using the MTF.TQS heuristic exceeds the optimal cost by merely 9.82%. To compare MTF.TQS and TR.TQS under identical settings consider the

results presented in Table 7.4. From this table, we observe that the expected cost under the TR.TQS heuristic exceeds the optimal cost by an amazingly low 5.42%.

For all the results reported in Tables 7.1-7.6, the TR.TQS heuristic consistently outperforms the MTF.TQS heuristic. While we believe the result is true for all possible query distributions, a general proof of this statement remains open. This result, if true, will be a generalization of the analogous relation that exists between the TR and MTF heuristics in the case when the queries consist of statistically independent, single record accesses[81].

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	6.1875	6.1941	5.6667	9.31
6	13.0765	13.0685	11.90	9.82
8	22.4534	22.4483	20.40	10.04
10	34.3542	34.3345	31.1667	10.16
12	48.7764	48.7275	44.20	10.24

Table 7.1: Results for the MTF.TQS heuristic, when the nature of the $\mathcal{P}$ vector is "linear" as per (7.46).

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.4676	5.4753	4.9949	9.62
6	10.8685	10.8672	9.90	9.77
8	17.9378	17.9513	16.40	9.46
10	26.7414	26.7154	24.5	9.04
12	37.1686	37.1474	34.2	8.62

Table 7.2: Results for the MTF.TQS heuristic, when the nature of the $\mathcal{P}$ vector is "Zipf's" as per (7.47).

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.4427	5.4503	4.9698	9.67
6	10.4930	10.4959	9.5735	9.63
8	17.0088	17.0219	15.6338	8.88
10	25.1252	25.1065	23.2475	8.00
12	34.7915	34.7816	32.4506	7.18

Table 7.3: Results for the MTF-TQS heuristic, when the nature of the $\mathcal{P}$ vector is “Exponential” as per (7.48).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	6.0219	5.6667	6.27
6	12.5452	11.90	5.42
8	21.338	20.40	4.60
10	32.4050	31.1667	3.97
12	45.7628	44.20	3.54

Table 7.4: Results for the TR-TQS heuristic, when the nature of the $\mathcal{P}$ vector is “linear” as per (7.46).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.3157	4.9949	6.42
6	10.4275	9.90	5.33
8	17.1139	16.40	4.35
10	25.4392	24.5	3.83
12	35.3430	34.2	3.34

Table 7.5: Results for the TR-TQS heuristic, when the nature of the $\mathcal{P}$ vector is “Zipf’s” as per (7.47).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.2864	4.9698	6.37
6	10.0519	9.5735	5.00
8	16.2221	15.6338	3.76
10	23.9537	23.2475	3.04
12	33.2091	32.4506	2.34

Table 7.6: Results for the TR.TQS heuristic, when the nature of the $\mathcal{P}$ vector is “Exponential” as per (7.48).

7.7 Summary

The problem of dynamically reorganizing a linear list in which the elements accessed are individually requested has been well studied. In this chapter, we have proposed a model of query generation in which any subset of the elements in the linear list can be requested. Examples of the occurrences of such queries in the context of distributed systems, and database systems have been presented. An exact analytical description of this scenario necessarily involves 2^N parameters, where N is the number of elements in the list. This being impractical, we suggest a simple (and hence approximate) model for the query generation strategy based on a vector $\mathcal{P}$, of dimension N . We have derived the asymptotic record access distribution of the serial query stream emerging from such a query generator. Statistical dependence between successive records in this stream has also been established.

Under the new model of query generation, the optimal list configuration has been shown to be the order corresponding to the descending order of the components of $\mathcal{P}$. Additionally, two heuristics MTF_TQS and TR_TQS capable of dealing with set queries have been proposed which are generalizations of the MTF and TR heuristics for singly-linked lists. The cost of the MTF_TQS heuristic for our query model has been shown to be no more than $\pi/2$ times the expected cost for the optimal list arrangement. Furthermore, it is also shown that MTF_TQS and TR_TQS are superior to the corresponding serialized heuristics, MTF_SQS and TR_SQS respectively. Experimental results involving the MTF_TQS and TR_TQS heuristics have been presented. The question of proving that TR_TQS is always superior than MTF_TQS under the proposed model of query generation remains open.

Chapter 8

Keyboard Optimization Problem

In the preceding chapters, we have dealt with adaptive learning algorithms for the design of adaptive data structures. We now present the last problem studied in this thesis, namely that of keyboard optimization. The reader will recall that this problem was presented informally in Section 1.2.2 of Chapter 1. Various applications of this problem (including data compression) were also identified there. We now proceed to state the problem formally and analyze its computational complexity.

8.1 Problem Statement and Notation

After introducing the notation we shall formally present the problem.

$\mathcal{A}$ is a finite alphabet of cardinality $|\mathcal{A}|$ and H , the dictionary, is a subset of the words over $\mathcal{A}^*$. Let $\mathbf{K}$ be a subset of the integers, where, with no loss of generality, $\mathbf{K} = \{1, 2, \dots, K\}$. With every element $i \in \mathbf{K}$, is associated the set C_i which is a subset of $\mathcal{A}$, where the C_i 's are mutually exclusive and collectively exhaustive. Whenever a string $X = x_1x_2\dots x_N \in H$ is to be transmitted (represented), instead of transmitting X in its virgin form we transmit the string $Y = y_1y_2\dots y_N$, where $y_i \in \mathbf{K}$ and $x_i \in C_{y_i}$.

The symbol Π will be used to represent the set (or partition) $\{C_i\}$. Thus $\Pi(X)$ will be the string Y , where if $X = x_1x_2\dots x_N \in H$, then $Y = y_1y_2\dots y_N$, with $y_i \in \mathbf{K}$ and $x_i \in C_{y_i}$. Also, for the sake of ease of expression we shall define H^Π to be the mapping of H due to Π . Thus,

$$H^\Pi = \{\Pi(X) | X \in H\} \quad (8.1)$$

Observe that the cardinalities of both H and H^Π need not be the same, and that $|H^\Pi| \leq |H|$.

Although every string Y now represents a set of strings in $\mathcal{A}^*$, it may or may not represent multiple strings in the original dictionary H primarily because H is but a subset of $\mathcal{A}^*$. To quantify the effectiveness of a particular partition Π being able to disambiguate the dictionary, we first define $\delta(X_0, H, \Pi)$ to be the number of collisions which a particular string $X_0 \in H$ possesses. Indeed, if X_0 is transformed to Y_0 and no other $X \in H$ is mapped into Y_0 , the number of collisions, $\delta(X_0, H, \Pi)$, for X_0 as per the partition Π is zero. However, if Z other strings in H are also mapped into Y_0 then the number of collisions, $\delta(X_0, H, \Pi)$ must be Z . Formally,

$$\delta(X_0, H, \Pi) = |\{X | \Pi(X) = \Pi(X_0); X, X_0 \in H; X \neq X_0\}| \quad (8.2)$$

Since the number of collisions per word has been explicitly defined, we can now define $\Delta(H, \Pi)$ to be the total ambiguity caused by a particular keyboard assignment (partition) Π . Using the definition for H^Π , $\Delta(H, \Pi)$ can be defined as:

$$\Delta(H, \Pi) = |H| - |H^\Pi|. \quad (8.3)$$

This is indeed the total number of collisions in the entire dictionary due to the assignment Π .

However, $\Delta(H, \Pi)$ can also be defined in terms of $\delta(X_0, H, \Pi)$ as below:

$$\Delta(H, \Pi) = \sum_{X_0 \in H} \frac{\delta(X_0, H, \Pi)}{\delta(X_0, H, \Pi) + 1} \quad (8.4)$$

The fact that (8.3) and (8.4) both represent identical expressions is not so obvious but can be verified by observing that if each individual $\delta(X_0, H, \Pi)$ is included in the summation, the quantity would be accounted for $(\delta(X_0, H, \Pi) + 1)$ times.

The problem studied in this chapter is one of computing the keyboard assignment, Π^* which minimizes $\Delta(H, \Pi)$. This problem is referred to as the Keyboard Optimization (OptKeyboard) problem and is formally stated below.

Problem OptKeyboard:**Input:**

Alphabet $\mathcal{A}$, Dictionary H , and an integer K .

Output:

Π^* , referred to as the optimal solution, i.e. $\Pi^* = \{C_i^* | 1 \leq i \leq K\}$, which is defined as follows. Let $\Pi = \{C_i | 1 \leq i \leq K\}$, where:

- (i) $\cup_{i=1}^K C_i = \mathcal{A}$
- (ii) $C_i \cap C_j = \phi \quad i \neq j$.

Then

$$\Pi^* = \text{Arg}_{\Pi}[\min(\Delta(H, \Pi))].$$

The decision version of the above optimization problem, will be called ApproxKeyboard and is formally stated below:

Problem ApproxKeyboard:**Input:**

Alphabet $\mathcal{A}$, Dictionary H , integers K and MaxCollisions.

Output:

Determine the existence of a partition Π^+ , referred to as an approximate solution, i.e. $\Pi^+ = \{C_i^+ | 1 \leq i \leq K\}$, which is defined as follows.

- (i) $\cup_{i=1}^K C_i^+ = \mathcal{A}$
- (ii) $C_i^+ \cap C_j^+ = \phi \quad i \neq j$.
- (iii) $\Delta(H, \Pi^+) \leq \text{MaxCollisions}$

In this chapter we shall prove the result that ApproxKeyboard is NP-Complete if the dictionary H is known *a priori*. We conjecture that the problem is provably exponentially hard when H is unknown but only the values of $\Delta(H, \Pi)$ are available to the program for all possible permutations, Π . As a consequence of the above result, the original problem OptKeyboard can then be easily seen to be an NP-hard problem.

After establishing the computational complexity of the problem, we discuss previous known solutions to the problem. Subsequently we propose a fast learning

automaton solution to the problem. In this chapter, experimental results demonstrating the power of the automaton will also be presented. We first justify the use of a finite dictionary in the keyboard optimization problem.

8.1.1 Dictionary Representation Mechanisms

Central to the Keyboard Optimization problem is the technique by which the dictionary H is represented. Indeed, the way by which H will be modeled depends on the application, and the set of possible models spans a whole spectrum of techniques including representing it as a finite dictionary [1, 27, 34, 35, 38, 39, 55, 85], representing H as being fully defined by bigrams and trigrams [21, 68, 83, 84, 85], or alternatively by representing H using binary positional n -grams [27, 85]. In this paper we shall present a disambiguation strategy for the finite dictionary model. The primary reasons why we have chosen to represent the dictionary as a finite set of word is explained in this subsection.

In most applications encountered in real life the dictionary used is finite. This is especially true in the case of telephone directories, and the vocabulary used by hospitalized handicapped individuals. It is also true if the device is used to communicate with a machine with a finite command set. Indeed, even in the case of written English text, various studies have been made which indicate that large proportions of the words used in English form a very small subset of the possible English words. In fact Dewey [17] (See [38] for results of string correction experiments done using these words) has compiled such a collection and claimed that this collection consisting of 1023 words comprise a very large proportion of written English text. Thus, in both string processing and string recognition it is not uncommon to represent the dictionary as a finite set of words, and using this model, string correction is often achieved by comparing the received (noisy) word with every single word in the dictionary using a similarity metric [27, 38, 39, 59, 83, 97, 101].

The advantages of using a finite dictionary in text recognition applications are many. First of all, the accuracy of the recognition is very high [27, 38, 39, 59]. Secondly, a noisy string is never recognized as a word which is not in the language,

and thus, the question of 'meaningless' decisions is nonexistent. Finally, the time complexity of the computation involved in the text recognition process is typically quadratic per word and is linear in the size of the dictionary. The quadratic complexity per word can be often decreased if the dictionary is modeled using a trie [39], and if the alphabet size is decreased [1, 83, 101].

When the dictionary becomes prohibitively large, it is represented using a statistical model [21, 68, 83, 84, 85, 94]. The most elementary model is the one in which only the unigram (single character) probabilities of the dictionary are required. A word in the dictionary is then modeled as a sequence of characters, where each character is independently drawn from a distribution referred to as the unigram distribution, and which is exactly the probabilities of the letter occurring in the language. A generalization of this is the one in which the bigrams of the language model the dictionary. A word in the dictionary is modeled as a sequence of symbols where two subsequent symbols x_i, x_{i+1} occur with the probability with which they occur in the language. The storage required for the bigrams is now quadratic with the alphabet size, and is independent of the size of the dictionary. However, if all the unigram and bigram probabilities have non-zero values, a string of length N can be one of $O(|\mathcal{A}|^N)$ possible strings, and the computation can be excessive if the search tree is not appropriately pruned.

One of the biggest disadvantages of modeling the dictionary using n -grams or positional n -grams [21, 68, 83, 84, 85, 94] is the fact that the word which is obtained after the string processing (correction) operation need not be a valid word in the language under consideration. Thus, all too often the word that has been obtained is "rejected", and this is especially the case when only bigrams are used to model the dictionary. The percentage of meaningless decisions decreases if the dictionary is modeled using n -grams where $n > 2$, but the memory required becomes excessive – it is of $O(|\mathcal{A}|^n)$ real numbers. Observe that representing a finite dictionary as a finite dictionary itself (i.e., as a set of words and not using a statistical model) is tantamount to specifying all the n -grams of the dictionary, and that for all values of n . If the latter is done, the problem of having meaningless words is nonexistent.

8.2 Complexity Of the Keyboard Optimization

In the previous section, we have presented the Keyboard optimization problem as a classical optimization problem, with a well defined objective function. In this section, we present the reduction which establishes the corresponding decision problem to be NP-complete. For the sake of concreteness we establish the result for the case when $K = 2$. The proof that ApproxKeyboard is NP-complete for all values of K can be easily generalized from the proof for the case when $K = 2$.

The problem we have on hand, can be related to the family of clustering problems studied by Brücker [8] and the MAX-CUT problem [23], which are known to be NP-complete. In [8], Brücker establishes a correspondence between the MAX-CUT problem and the 2-Clustering problem, in which the set of nodes is to be split into two clusters, according to a given criterion function. A derivative of the former (i.e. the MAX-CUT problem) in which all the edges of the graph involved have only unit weights was shown to be NP-complete in [24], and thus, following straightforward techniques analogous to Brücker's, it is easy to see that the "Simplified 2-clustering" problem (SIMP2C) is also NP-complete. Since our proof involves SIMP2C, we formally state the result which we shall use later.

Lemma 8.0 (SIMP2C) *Given a graph $G = (V, E)$ with only unit edge weights (i.e. $w(e) = 1$ for each $e \in E$), and an integer M , the problem of partitioning the set of vertices V into two disjoint sets V_1 and V_2 such that $\psi(V_1, V_2) \leq M$, is NP-complete, where:*

$$\psi(V_1, V_2) = \sum_{l=1}^2 \sum_{a,b \in V_l, (a,b) \in E} w(\langle a, b \rangle) \quad (8.5)$$

The fundamental result of this chapter is stated in the following theorem.

Theorem 8.1 *ApproxKeyboard is NP-complete.*

Proof:

In order to prove the problem to be NP-complete, we have to establish the following [23]:

- (i) ApproxKeyboard is in the class NP.
- (ii) Some known NP-complete problem reduces to ApproxKeyboard (or a restricted instance of it).

To prove the first contention, i.e. that our problem belongs to the class NP is rather straightforward. The “non-deterministic” part of our algorithm need only guess the correct partition; the verification merely consists of the following steps:

1. Encoding each word in the dictionary H .
2. Counting the collisions (possibly after the sorting the encoded dictionary H^Π).
3. Verifying that the number of collisions is less than or equal to MaxCollisions.

Obviously steps 1 to 3 in the verification process can be accomplished in polynomial time.

To prove the second contention of the Theorem, we shall use the SIMP2C problem and show that an instance of it reduces to the ApproxKeyboard problem. The reduction between SIMP2C and ApproxKeyboard is formally presented in Algorithm Reduce in Program 8.1. Observe that in the algorithm we have used the notation that if $a_p \in \mathcal{A}$, a_p^M is the string consisting of M consecutive occurrences of a_p .

Algorithm Reduce**Input:**

An instance of SIMP2C which is a graph $G = (V, E)$, where $V = \{a_1, a_2, \dots, a_R\}$, and the edge weights are unity for the edges in E and an integer M .

Output:

An instance of ApproxKeyBoard with the following parameters: A set of characters representing the alphabet $\mathcal{A}$, the dictionary H , and the integer MaxCollisions.

Method:

```

MaxCollisions :=  $M$ 
 $\mathcal{A} := V$  {Step 1: Choose alphabet  $\mathcal{A}$ }
 $m := 0$ 
 $H := \{\}$ 
For each edge  $e = \langle a, b \rangle \in E$  do
     $m := m + 1$ 
    Word1 :=  $a^m b^m$ 
    Word2 :=  $b^m a^m$ 
     $H := H \cup \{\text{Word1}, \text{Word2}\}$ 
endfor {Step 2: Form Dictionary  $H$ }

```

End Algorithm Reduce

Program 8.1: Algorithm Reduce: Reduces SIMP2C to ApproxKeyBoard.

To illustrate the operation of Algorithm Reduce, an example has been presented in Figure 8.1. The input graph consists of 6 nodes and 8 edges as depicted in Figure 8.1. When Algorithm Reduce is applied to the graph shown in Figure 8.1, the outputs from Algorithm Reduce which serve as an input to ApproxKeyBoard are:

$$\mathcal{A} = \{a,b,c,d,e,f\}$$

$$H = \{ ab,ba, a^2c^2, c^2a^2, a^3d^3, d^3a^3, b^4d^4, d^4b^4, b^5f^5, f^5b^5, \\ c^6f^6, f^6c^6, d^7e^7, e^7d^7, f^8e^8, e^8f^8 \}$$

Figure 8.1: Input graph $G = (V, E)$.

We now analyze the complexity of Algorithm Reduce. Step 1 of the transformation takes time $O(n)$ where $n = |V|$. Note that in Step 2 the length of the words increase linearly with every edge $e \in E$ which is examined and hence the cost is $O(1 + 2 + \dots + |E|)$ or $O(|E|^2)$. Hence the total reduction can be achieved at a total cost of $O(n + |E|^2)$, which is polynomial in the input size.

It now remains to be proved that the sequence of steps mentioned above, indeed constitutes a **transformation**, i.e. that this sequence preserves the one-to-one correspondence between the solutions of the two problems. We observe the following properties of the Dictionary H created by Algorithm Reduce.

- (i) A pair of letters, say (α, β) appear in **exactly two words** in the entire dictionary. These words have the form $\alpha^m \beta^m$ and $\beta^m \alpha^m$. We shall refer to these words as $\text{Word}(\alpha, \beta)$ and $\text{Word}(\beta, \alpha)$ respectively.

(ii) There are exactly two words of the same length in the entire dictionary.

As a consequence of the above properties, the word pairs of the form $\alpha^m \beta^m$ and $\beta^m \alpha^m$ can contribute to exactly one collision, and that, only when the letters α and β are placed on the same key. These words contribute to no collisions otherwise. To be more explicit,

$$\delta(\alpha^m \beta^m, H, \Pi) = \delta(\beta^m \alpha^m, H, \Pi) = \begin{cases} 1 & \text{if } \alpha, \beta \in C_i, \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

Consider the expression:

$$\frac{\delta(\alpha^m \beta^m, H, \Pi)}{\delta(\alpha^m \beta^m, H, \Pi) + 1} + \frac{\delta(\beta^m \alpha^m, H, \Pi)}{\delta(\beta^m \alpha^m, H, \Pi) + 1}. \quad (8.6)$$

First of all, observe that the expression is meaningless if $\alpha^m \beta^m$ and $\beta^m \alpha^m$ are not in H . In other words, the expression is meaningless if the instance of SIMP2C did not possess the edge $\langle \alpha, \beta \rangle$. Notice that in the meaningful case, $w(\langle \alpha, \beta \rangle) = 1$. Both the terms in expression (8.6) are either 0 or $\frac{1}{2}$ simultaneously. They are zero if α and β are on different keys. Thus,

$$\frac{\delta(\alpha^m \beta^m, H, \Pi)}{\delta(\alpha^m \beta^m, H, \Pi) + 1} + \frac{\delta(\beta^m \alpha^m, H, \Pi)}{\delta(\beta^m \alpha^m, H, \Pi) + 1} = \begin{cases} 1 & \text{if } \alpha, \beta \in C_i \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

Or equivalently,

$$\begin{aligned} & \frac{\delta(\alpha^m \beta^m, H, \Pi)}{\delta(\alpha^m \beta^m, H, \Pi) + 1} + \frac{\delta(\beta^m \alpha^m, H, \Pi)}{\delta(\beta^m \alpha^m, H, \Pi) + 1} \\ &= \begin{cases} w(\langle \alpha, \beta \rangle) & \text{if } \alpha, \beta \in C_i \text{ for some } i \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (8.7)$$

It now remains to be proved that the objective functions for the 2-Clustering and the reduced ApproxKeyboard problems are identical. The total number of collisions in the dictionary, $\Delta(H, \Pi)$ can be written down using (8.4), as:

$$\begin{aligned} \Delta(H, \Pi) &= \sum_{X_0 \in H} \frac{\delta(X_0, H, \Pi)}{\delta(X_0, H, \Pi) + 1}, \\ &= \sum_{\langle \alpha, \beta \rangle \in E} \left\{ \frac{\delta(\text{Word}(\alpha, \beta), H, \Pi)}{\delta(\text{Word}(\alpha, \beta), H, \Pi) + 1} + \frac{\delta(\text{Word}(\beta, \alpha), H, \Pi)}{\delta(\text{Word}(\beta, \alpha), H, \Pi) + 1} \right\} \end{aligned}$$

$$\begin{aligned}
&= \sum_{\substack{\alpha, \beta \text{ in same cluster} \\ \{\alpha, \beta\} \in E}} w(\langle \alpha, \beta \rangle) \quad (\text{from (8.7)}) \\
&= \sum_{l=1}^2 \sum_{\substack{\alpha, \beta \text{ in cluster } V_l \\ \{\alpha, \beta\} \in E}} w(\langle \alpha, \beta \rangle). \tag{8.8}
\end{aligned}$$

The reader will notice that (8.8) is exactly the objective function for the underlying 2-Clustering problem, which was specified by (8.5). Hence a solution to the ApproxKeyboard problem leads to a solution of the original instance of the 2-Clustering problem as well and the theorem is proved. $\square$

It is clear that our reduction is completely general and does not depend on the value of K . Thus the ApproxKeyboard problem is NP-complete for all values of K .

Observe that the above proof only established the NP-completeness of ApproxKeyboard with the implicit assumption that the dictionary H was *known* to the algorithm which attempts to solve the problem. If we work with the assumption that the dictionary H was *not known* to the algorithm, but rather that the algorithm was presented with a function or a table which returned the number of collisions for a specified partition Π , the problem would become much more difficult. We conjecture that when the problem is posed in this form, the computational complexity is *provably exponential*.

8.3 Previous Solutions to OptKeyboard

Since the Keyboard Optimization Problem is NP-hard, we are interested in finding fast (approximate) solutions to it. The solutions which are presented here converge to a solution which is, hopefully, close enough to the optimal one. In this section we shall present the only solution reported in the literature and in the subsequent sections we shall present a new learning automaton solution which is superior to the former.

The only reported solution to the keyboard optimization problem is an evolutionary method [52, 53, 65] based on a technique described by De Jong [15]. The

latter is an adaptive algorithm based on genetic reproduction. It is represented using the standard feedback loop of classical control theory. This adaptive system is formally represented by De Jong as:

- (i) A set E of environments in which the complex process must function.
- (ii) A set I of all allowable control inputs which are available to an adaptive strategy for modifying the behavior of the complex process.
- (iii) A process performance measure $u : E \times I \mapsto \mathfrak{R}$, where $\mathfrak{R}$ represents the set of reals.
- (iv) A feedback function $f : E \times I \mapsto \mathfrak{R}^n$ providing the adaptive strategy with information about the process being controlled.
- (v) A set S of adaptive strategies each of which attempts to use the knowledge obtained by the feedback interaction to improve the performance of the process.

De Jong's technique assumes that the adaptive system is operating in discrete time and that it has no *a priori* information about the set of environments. Using these assumptions De Jong presented an algorithm known as the reproductive plan, which belongs to an entire variety of genetic-based adaptive strategies. In the reproductive plan each point or each element $c \in I$ is considered to be an individual organism. This organism is further represented uniquely by a string generated from some alphabet, where this string is the equivalent of the DNA of a real organism, and with the additional constraint that specific positions on the string take on different values exactly as genes take on different alleles. The performance measure is now defined as the fitness of the organism in a particular environment, and a more fit organism relates directly to a better control input.

The reproduction plan simply starts off by randomly creating a number of organisms which constitute the first generation. Each organism is assigned a fitness according to the performance measure and is allowed to reproduce. Various reproductive techniques are permitted and these include the mutative and inversive techniques. Independent of the reproductive technique used a superior (more fit)

organism always possesses a higher probability of generating an offspring than an inferior (less fit) one. The number of offsprings which an organism can have may be defined by its fitness factor divided by the average fitness of the population as follows:

$$O(c) = \frac{u(c, e)}{\sum_{c \in I} u(c, e)/N}, \quad \text{where } N \text{ is the population size.} \quad (8.9)$$

In general, an average organism will have one offspring, a more fit organism will have more offsprings and a less fit organism might be barren. It should be noted that $O(c)$ is a real number, and thus, if in a particular reproductive mechanism $O(c) = 1.23$, this could imply that the organism would certainly have one offspring, and furthermore that the second offspring will be created with a probability of 0.23. After all the organisms in a particular generation have reproduced, the offsprings themselves form the next generation and the process continues, while the old generation dies off. As time progresses the population should become more fit on the average since the better organisms will take over a larger portion of the population. The latter statement, of course, assumes that the children of a more fit organism will on the average be more fit than the children of a less fit individual. Although this assumption is hard to prove, it is intrinsically tied in with the genetic DNA representation and is thus probably justified.

One of the simpler forms of reproduction involves crossover. For every organism in the population a mate is chosen randomly (or otherwise). The DNA strings of the two individuals are now cut and spliced together at some mutual point. For example, if the length of the DNA is L , a random point $p, 1 \leq p < L$ is picked on the string. The child's DNA string will be formed using the first parent's DNA string positions from 1 through p spliced with the other parent's DNA string from positions $p + 1$ through L . Observe, however, that a crossover operation by itself is not able to create individuals which are radically different from their parents. For example, if every organism in the population has the element α in the first position, a child produced as a result of crossover will also always have the same element in its first position. In order to overcome this limitation a second factor is

introduced into the reproductive process, namely, mutation. A mutation operation is able to independently alter one or more of the gene values, and this is typically achieved randomly. The probability of a mutation occurring is usually small but this operation nevertheless guarantees that the set of organisms generated will eventually span the whole space.

The adaptation of this approach to the keyboard disambiguation problem presented earlier is quite simple. The set of environments E will be represented by a set of dictionaries, where one specific environment is a particular dictionary H . The set I now represents a set of all possible keyboards. The process performance function u is identically equal to the function $\Delta(H, \Pi)$. Finally, the set S of adaptive strategies contains only one adaptive strategy - that of reproduction involving the crossover and mutation operations. More explicitly, the DNA of the organism will be represented by a string of length $|\mathcal{A}|$, where $\mathcal{A}$ is the finite alphabet. In each position the index of the key on which the character currently appears will be used to represent the organism's (keyboard's) DNA string, and thus if 'a' is assigned key 3 it is meant to represent that 'a' $\in C_3$.

Levine and Minneman used this fundamental principle to minimize keyboard ambiguities[64, 65, 52]. However, as opposed to our measure, $\Delta(H, \Pi)$, which evaluated the ambiguity of a particular keyboard assignment, their solution used an evaluation function based on a bigram metric. Their procedure as reported does not result in a solution which monotonically minimizes the number of collisions encountered. Monotonicity can however be attained by making a simple modification to the method suggested by Levine and Minneman. It is this variant which was used as a benchmark to assess the performance of our learning automaton solution.

8.4 Automata Solutions To OptKeyboard

Our solution to the problem involves the use of stochastic learning automata. Learning automata have been used in the literature to model biological learning systems and also to learn the optimal action which an environment offers. The learning is

achieved by actually interacting with the environment and processing its responses to the actions that are chosen. Such automata have various applications such as parameter optimization, statistical decision making and telephone routing [47, 66, 67]. Since the literature on learning automata is extensive, we refer the reader to a paper by Narendra and Thathachar [67] and two excellent books on the field by Lakshminivarahan [47] and Narendra and Thathachar[66] for a review of the families and applications of learning automata.

The learning process of an automaton can be described as follows: The automaton is offered a set of actions by the environment with which it interacts, and it is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized by the environment with a certain probability. A learning automaton is one which learns the optimal action, which is the action which has the minimum penalty probability. Hopefully, the automaton will eventually choose this action more frequently than other actions.

Stochastic learning automata can be classified into two main classes : (a) Fixed Structure Stochastic Automata (FSSA) and (b) automata whose structure evolve with time. Some examples of the former type are the Tsetlin, Krinsky and Krylov automata [67, 89]. Although the latter automata are called variable structure stochastic automata, because their transition and output matrices are time varying, in practice, they are merely defined in terms of action probability updating rules [47, 67]. Although throughout this chapter we will only be considering FSSA, automata with a variable structure are generally much faster in their convergence.

8.4.1 Fundamentals of Learning Automata

A FSSA is a quintuple $(\alpha, \Phi, \beta, F, G)$ where:

- (i) $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_R\}$ is the set of actions that it must choose from.
- (ii) Φ is its set of states.
- (iii) $\beta = \{0, 1\}$ is its set of inputs. The input '1' represents a penalty.

- (iv) F is a map from $\Phi \times \beta$ to Φ . It defines the transition of the state of the automaton on receiving an input. F may be stochastic.
- (v) G is a map from Φ to α , and determines the action taken by the automaton if it is in a given state. With no loss of generality, G is deterministic [66].

The selected action serves as the input to the environment which gives out a response $\beta(n)$ at time n . $\beta(n)$ is an element of $\beta = \{0, 1\}$ and is the response of the environment which is fed back to the automaton. The environment penalizes the automaton with the probability c_i , where,

$$c_i = Pr[\beta(n) = 1 | \alpha(n) = \alpha_i] \quad i = 1, 2, \dots, R.$$

Thus the environment characteristics are specified by the set of penalty probabilities $\{c_i | 1 \leq i \leq R\}$. On the basis of the response $\beta(n)$ the state of the automaton $\phi(n)$ is updated and a new action is chosen at the time instant $(n + 1)$.

The $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action which presents it with the minimum penalty response in an expected sense. Note that if c_L is the minimum penalty probability, and if:

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

the solution:

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

achieves this result. Automata are designed with this solution in view.

With no *a priori* information, the automaton chooses the actions with equal probability. The expected penalty is thus initially M_0 , the mean of the penalty probabilities.

In the case under consideration, if the set of actions is the set of all possible partitions of $\mathcal{A}$, the penalty function associated with the actions is a deterministic function. For a particular partition Π , this function is indeed the associated number of collisions $\Delta(H, \Pi)$.

An automaton is said to learn **Expediently** if, as time tends towards 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 learn **Absolutely Expediently** if for all n , $E[M(n+1)] < E[M(n)]$. We say that the automaton is **Almost Absolutely Expedient** if the latter inequality not strict, i.e., if for all n , $E[M(n+1)] \leq E[M(n)]$.

The automaton is said to be **optimal** if $E[M(n)]$ asymptotically equals the minimum penalty. It is ϵ -optimal if, in the limit, $E[M(n)]$ can be made as close to this minimum penalty as desired by the user. This is usually achieved by a suitable choice of some parameter of the automaton, for example, the number of states of the automaton.

8.4.2 The Krinsky Automaton

The reward characteristics of the automaton solution which we shall present is akin to that of a well known automaton due to Krinsky [89]. The K -action Krinsky automaton, $Z_{KN,K}$, has KN states $\{\phi_1, \phi_2, \dots, \phi_{KN}\}$ and chooses one of the K actions $\{\alpha_1, \alpha_2, \dots, \alpha_K\}$. N is called the “Depth” of the automaton. Explicitly, $Z_{KN,K}$ is defined as follows:

$$Z_{KN,K} = (\{\phi_1, \phi_2, \dots, \phi_{KN}\}, \{\alpha_1, \alpha_2, \dots, \alpha_K\}, \{0, 1\}, Z(\cdot, \cdot), G(\cdot)).$$

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

$$G(\phi_i) = \alpha_j \quad \text{if} \quad (j-1)N + 1 \leq i \leq jN. \quad (8.10)$$

Observe that since 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., the states of the automaton are automatically partitioned into groups, each group representing an action.

The $Z(\cdot, \cdot)$ map is deterministic and is described as below:

1. If $\beta = 0$ (it gets a favorable response), the Z map requires that the automaton goes directly towards the most internal state corresponding to that action — $\phi_{(j-1)N+1}$ for action α_j . Explicitly,

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

2. If $\beta = 1$ (it gets an unfavorable response), the automaton moves towards the boundary state — ϕ_{jN} if α_j is the action chosen — one step at a time. If it is in the boundary state for the action, it moves to the boundary state for the next action. Thus, for all $1 \leq j \leq K$:

$$Z(\phi_i, 1) = \begin{cases} \phi_{i+1} & \text{if } (j-1)N + 1 \leq i < jN \\ \phi_{(j+1)N \bmod KN} & \text{if } i = jN, \quad j \neq K-1 \\ \phi_{KN} & \text{if } i = (K-1)N \end{cases}$$

We now state the following theorem which was proved two decades ago [89].

Theorem 8.2 *The $Z_{KN,K}$ automaton is ϵ -optimal in all random environments.*

8.4.3 The Keyboard Learning Automaton

Closely related to the problem which we are studying is the Equi-partitioning problem which can be described as follows. Let $\{A_1, A_2, \dots, A_W\}$ be a set of W objects. In the Equi-partitioning problem we intend to partition this set into R classes $\{P_1, P_2, \dots, P_R\}$ of equal size. Further, we intend to partition them in such a way that the objects that are accessed (used) more frequently together lie in the same class. Note that to render the problem non-trivial, we assume that these joint access probabilities are unknown. Applications of the object partitioning problem are found in information retrieval, attribute partitioning, record clustering and in distributed file allocation.

Various adaptive solutions to the Equi-partitioning problem have suggested in the literature. The best known one in the early 70's was due to Hammer *et. al.* [28] which was essentially a hill-climbing solution to the underlying optimization problem. Yu *et. al.* [102] proposed a superior algorithm to solve this problem called the Basic Adaptive Method. The best known solution of the Equi-partitioning problem uses learning automata [70]. In [70], a new automaton called the Object Migrating Automaton was designed by Oommen *et. al.* which converged an order

of magnitude faster than the Basic Adaptive Method. The reader who is aware of the work done in Equi-partitioning [70] will find the following solution to the OptKeyboard problem to be an interesting generalization of the Object Migrating Automaton.

The learning automaton solution presented here is called the Keyboard Learning Automaton (KLA). We define KLA as a 6-tuple as below:

$$(\mathcal{A}, \{\phi_1, \phi_2, \dots, \phi_{KN}\}, \{\alpha_1, \alpha_2, \dots, \alpha_K\}, \{0, 1\}, Q(.,.), G(.)),$$

where,

- (i) $\mathcal{A}$ is the finite alphabet.
- (ii) $\{\phi_1, \phi_2, \dots, \phi_{KN}\}$ is the set of states.
- (iii) $\{\alpha_1, \alpha_2, \dots, \alpha_K\}$ is the set of K actions, each representing a certain class into which the elements of $\mathcal{A}$ must fall.
- (iv) $\{0, 1\}$ are the inputs to the automaton. As before '0' represents rewarding the automaton and '1' represents penalizing it.
- (v) Q , the transition function is quite involved and will be explained in detail presently.
- (vi) The function G is identical to the one described in (8.10). Thus, for each action α_k , there is a set of states $\{\phi_{(k-1)N+1}, \dots, \phi_{kN}\}$, where N is the depth of memory and $1 \leq k \leq K$. With no loss of generality, we assume $\phi_{(k-1)N+1}$ to be the most internal state of action α_k and ϕ_{kN} be the boundary state.

As opposed to the automata customarily studied in the literature, in this case, we have all the characters of $\mathcal{A}$ moving around in the states of the machine. If the character $a_i \in \mathcal{A}$ is in action α_k , it signifies that this character will be in C_k , the set numbered k . Observe that if the states occupied by the characters are given, the actions chosen by the characters of $\mathcal{A}$ can be trivially obtained using (8.10), and these specify the partition currently chosen by the automaton.

Let $\xi_a(n)$ be the index of the state occupied by $a \in \mathcal{A}$ at time n . Also let $\Xi(n) = \{\xi_a(n) | a \in \mathcal{A}\}$. Based on $\Xi(n)$, let $\Pi(n)$ be the current partition which the automaton decides. Observe that, as mentioned above, this partition is trivially obtained by repeatedly using (8.10) to the various elements of $\Xi(n)$ to determine the action chosen by each character $a \in \mathcal{A}$. Using this notation we shall now describe the transition map of the KLA. However, in what follows, in the interest of brevity, we shall omit the reference to the time instant n .

Initially, the KLA begins its learning process by starting from an arbitrary partition $\Pi(0)$, and this initial partition may be obtained randomly or otherwise. Thereafter the following process continues until the user is satisfied with the partition. Based on the current partition, Π the mapping of H due to Π , H^Π is computed. The words in H are now processed in a sequential manner. On processing a word $X \in H$, the mapped dictionary, H^Π , is searched and the number of collisions $\delta(X_0, H, \Pi)$ is computed using (8.2). If the number of collisions is zero, obviously X can be encoded using Π uniquely and unambiguously. In this case the partition Π is rewarded. Otherwise, the representation is ambiguous and the partitioning Π is penalized. The operations involved with rewarding and penalizing the partitioning Π are described below.

The Reward operation is quite simple. Every letter in the processed word, X , is moved to the internal state of its corresponding action as per the transition map of the Krinsky automaton (see Figure 8.2).

As opposed to this, the Penalty operation of penalizing a partition Π is more intricate and it involves penalizing a set of characters called the set of colliding characters. This set is essentially the set of characters in X which are “responsible” for the collisions. To explicitly define this set, let $X = x_1x_2 \dots x_N$ and $W = w_1w_2 \dots w_N$ be two words in H such that $\Pi(X) = \Pi(W)$. Then T , the set of colliding characters in X is defined as follows:

$$T = \bigcup_{W \in H, \Pi(W) = \Pi(X)} \{x_i | x_i \neq w_i\}$$

For example, if $X = \text{"ace"}$, and it collides with the strings W and V , where $W = \text{"age"}$ and $V = \text{"ago"}$, T , the set of colliding characters in X is $\{ 'c', 'e' \}$. The elements of T will be moved as a consequence of the penalty operation.

Once the set T is identified, the individual characters in it are each moved towards the boundary state from one state to its adjacent one until at least one character is at the boundary state. One of the colliding characters in the boundary state is randomly chosen and moved to the boundary state of another randomly chosen action with an attempt to ameliorate the present partitioning $\Pi(n)$ (see Figure 8.3). The total number of collisions for the current partitioning is evaluated, and if the new partitioning causes less collisions or an equal number of collisions than the original partitioning, this becomes the current partitioning, $\Pi(n + 1)$.

To catalyze the convergence it is sometimes advantageous to use *a priori* information and specify additional restrictions on the set of possible partitions. For example, one such restriction is to constrain every key to have no more than four characters and no less than two characters. In such a case, in the event of a penalizing operation, when the colliding character moves to a new action the other characters associated with the new action may have to be moved if this constraint is violated. Thus, if the user imposes such a restriction and this restriction is violated on moving the colliding character, a random character associated with the new action is swapped so as to move to the boundary state of the original action of the colliding character (see Figure 8.4).

Figure 8.2: KLA with unrestricted characters per key. KLA processes the word $X = x_1 x_2 \dots x_N$ and every character x_i of X is rewarded. In the figure, the character x_i is in the action α_k .

Figure 8.3: This figure shows the actions chosen by KLA when the word $X = x_1x_2 \dots x_N$ is processed. The colliding characters of X (x_i and x_j in this figure) need to be punished. Initially, x_i and x_j are in actions α_k and α_l respectively. x_i moves to a random action α_m and to its boundary state if the number of collisions decreases. The number of characters per key is unrestricted.

Figure 8.4: This figure shows the actions chosen by KLA when the word $X = x_1x_2 \dots x_N$ is processed. The colliding characters of X (x_i and x_j in this figure) need to be punished. Initially, x_i and x_j are in actions α_k and α_l respectively. x_i moves to a random action α_m and to its boundary state if the number of collisions decreases. α is a random character moved to α_k to satisfy the user's restriction (the number of characters per key).

Our automaton based solution to OptKeyboard which was informally described above is formally presented as Algorithm KLA in Program 8.2. Note that shall merely manipulate the indices of the states occupied by the various characters in $\mathcal{A}$.

Algorithm KLA

Notation:

The KLA has N states per action, and ξ_a is the index of the state associated with the character $a \in \mathcal{A}$. Thus ξ_a satisfies $1 \leq \xi_a \leq KN$, and furthermore if $a \in \mathcal{A}$ is assigned to action α_k , $(k-1)N+1 \leq \xi_a \leq kN$. Note that all states of the form ϕ_{kN} are boundary states. The j^{th} character of a given word X is represented by x_j , where $1 \leq j \leq |X|$. Given the set $\Xi = \{\xi_a | a \in \mathcal{A}\}$ the partition chosen by the KLA is computed trivially using (8.10). We assume that the KLA has access to a function ComputePartition which has as its input the set Ξ and yields for its output the partition Π . We assume that the number of characters per key is unrestricted by the user.

Input:

- (i) An initial partition Π . The characters are initially placed on the boundary states of the actions of the KLA. Thus if $a \in \mathcal{A}$ is assigned to the key k , then ξ_a is assigned the value kN .
- (ii) A finite dictionary H .
- (iii) A stream of words from H . The words are processed one at a time.

Output:

A final partition Π^* obtained as a result of the learning process.

Assumptions:

- (i) The algorithm has access to functions FindNumberOfMatches and TotalCollisions. Given a word X and a set of character assignments, Ξ , FindNumberOfMatches yields $\delta(X, H, \Pi)$. TotalCollisions computes $\Delta(H, \Pi)$ for Ξ .
- (ii) The algorithm has access to a function CollidingCharacters, which has as its parameters a given string X , the dictionary H , and the current states occupied by the characters, Ξ . The function first computes H^Π and subsequently evaluates the set of words $\{X_j\}$ for which $\Pi(X_j) = \Pi(X)$. The set of characters in X which cause the collisions with the various strings in $\{X_j\}$ are then evaluated.

MainAlgorithm**Begin**

```

    GetInitialAssignment ( $\Xi$ ) /*  $\Xi$  is the current state assignments */
     $\Pi := \text{ComputePartition} (\Xi)$  /*  $\Pi$  is obtained using (8.10) */
    Repeat
        ReadWord( $X$ )
        Matches := FindNumberOfMatches ( $X, H, \Pi$ )
        If (Matches = 0) Then Ignore  $X$  /*  $X$  is not an element of  $H$  */
        Else
            If (Matches = 1) Then /*  $X$  is uniquely mapped by  $\Pi$  */
                Reward ( $X, \Xi$ )
            Else
                Punish ( $X, H, \Xi$ )
            EndIf
        EndIf
         $\Pi := \text{ComputePartition} (\Xi)$ 
    Until Satisfied
     $\Pi^* := \Pi$ 

```

End MainAlgorithm**Procedure Reward (X, Ξ)**

```

    For  $j := 1$  to  $|X|$  Do /* process all characters in  $X$  */
        If  $(\xi_{x_j} \bmod N) \neq 1$  Then
             $\xi_{x_j} := (\xi_{x_j} \text{ div } N) * N + 1$  /* Move  $\xi_{x_j}$  to the most internal state */
        EndFor

```

End Procedure Reward**Procedure Punish (X, H, Ξ)**

```

    Previous $\Delta := \text{TotalCollisions}(H, \Xi)$ 
     $T := \text{CollidingCharacters} (X, H, \Xi)$ 
    Repeat
        For all  $a \in T$  Do /* Do for all colliding characters in  $X$  */
            If  $(\xi_a \bmod N) \neq 0$  Then /*  $a$  is not in boundary state */
                 $\xi_a = \xi_a + 1$  /* Move  $a$  towards boundary state */
            EndFor /* Till at least one element is at the boundary */
    Until (some elements of  $T$  are in their boundary states).
     $b := \text{Random element in } T \text{ in boundary state.}$ 
    Repeat /*  $b$  in boundary state is to be moved */
         $k := \text{Random}(1, K)$ 

```



```

         $\xi_b := kN$  /* Move b to boundary state of  $\alpha_k$  */
    Until (TotalCollisions ( $H, \Xi$ )  $\leq$  Previous $\Delta$ ).
End Procedure Punish

```

END Algorithm KLA

Program 8.2: Algorithm KLA.

The learning property of Algorithm KLA now follows.

Theorem 8.3 *Algorithm KLA is both Expedient and Almost Absolutely Expedient.*

Proof:

Obvious. □

Remarks:

1. In the above description we have assumed that the user has no additional constraint on the number of characters per key. As mentioned earlier, often, to catalyze the convergence of the automaton it is advantageous to specify additional restrictions on the set of possible partitions. If the user requires that such a restriction is necessary, in the Procedure Punish, whenever the constraint is not satisfied the character in α_k closest to the boundary is moved to the boundary state of the action from which the colliding character is moved (see Figure 8.4). This is analogous to the operation used to moving objects in the Object Migrating Automaton for the Equi-partitioning problem [70].
2. Observe that the result proved above, that the automaton is Almost Absolutely Expedient, is weaker than we would have liked had we been able to design a machine which is Absolutely Expedient. However, considering the fact the problem is inherently NP-hard, this solution, although not as strong as we would like, is both fast and accurate. Indeed, on the basis of our experimental

results we conjecture that the machine is also ϵ -optimal viewed from the point of the current (discrete) criterion function.

An example will help clarify the operation of Algorithm KLA.

Example 1

Let $\mathcal{A}$ be the English alphabet and the dictionary H consist of eight words and be:

$$H = \{\text{ace, ale, ago, each, eels, ape, big, dig}\}.$$

Also let $\Pi(n)$ be the partition:

$$\Pi(n) = ((\text{aeo}), (\text{cgl}), (\text{bd}), (\text{fgr}), (\text{km}), (\text{inpz}), (\text{tuy}), (\text{hsv}), (\text{jwx})).$$

Then, the mapping of the various words in H is given by the set $\{121, 121, 121, 1128, 1128, 161, 362, 362\}$. Notice that if we extract the duplicate entries in the latter set, we get H^Π to be: $H^\Pi = \{121, 1128, 161, 362\}$.

Thus $\Delta(H, \Pi)$ has the value 4.

On processing the word $X = \text{"ace"}$ it is observed that there are two collisions, and that the set of colliding characters are $\{\text{'c'}, 'e'}\}$. Let us assume that 'c' is closer to the boundary state than 'e'. Since no restrictions are placed on the number of character per key, the character 'c' is now moved to a random action. If the random action picked is 6, the character 'c' will now be moved to action α_6 . Effectively, the character 'c' simply moves from key 2 to key 6 to form a temporary partition Π' , where,

$$\Pi' = ((\text{aeo}), (\text{gl}), (\text{bd}), (\text{fgr}), (\text{km}), (\text{cinpz}), (\text{tuy}), (\text{hsv}), (\text{jwx})).$$

Observe that $H^{\Pi'}$ is the set: $\{161, 121, 1168, 1128, 161, 362\}$. Since $\Delta(H, \Pi')$ is 2, the partition $\Pi(n+1)$ is assigned the value Π' .

If the user had placed an additional restriction that the number of characters per key had to be between 2 and 4 inclusive, on moving 'c' to α_6 , a random character is chosen from the characters choosing $\alpha_6 - \{\text{'i'}, 'n', 'p', 'z'}\}$. If 'p' was this random character, 'p' is moved to α_2 , the original action of 'c' to yield the partitioning $\Pi(n+1)$ to be:

$$\Pi(n+1) = ((\text{aeo}), (\text{glp}), (\text{bd}), (\text{fgr}), (\text{km}), (\text{cinz}), (\text{tuy}), (\text{hsv}), (\text{jwx})).$$

8.5 Experimental Results

The Evolutionary method described (EVOL-M), and the KLA were compared experimentally primarily to compare their rates of convergence and to evaluate their relative efficiencies. In all the cases the alphabet used consisted of the English alphabet. The dictionary used consisted of about 1100 words comprising primarily of the 1023 most common English words [17] and various words encountered in Computer Science.

Various simulations were conducted to evaluate the performance of the algorithms under a variety of constraints. In each simulation ten parallel experiments were conducted so that an accurate ensemble average of the results could be obtained. The size of the set, K was also a parameter to the programs, but in all the results we report below we assume that K is 9. Also in each case the original keyboard was chosen randomly.

The experiments were primarily of two types. In the first type of experiments the keyboard was to be learnt without any additional user's constraints. Unlike these, in the second it was assumed that every key on keyboard was constrained to have between two and four characters (inclusive). The results of the experiments are explained below, and a catalogue of these results is given in Table 8.1.

In all our experiments, the dictionary used was of size 1081, which included the 1023 most common English words[17]. The "depth" parameter for the KLA (i.e. N) was set to be 10. In Table 8.1, EVOL-M is the Optimized version of Levine and Minneman's scheme [52, 53, 65]. Also, KLA-U refers to the variant of KLA in which the number of characters per key is unrestricted. KLA-R represents the scheme in which the number of characters per key is between 2 and 4 inclusive.

Table 8.1 reports the average number of collisions in the dictionary as a function of the number of keyboards that are examined for various algorithms. For the case of the KLA in which the number of characters per key was unrestricted, the average

initial number of collisions was 65.5. This figure fell to 28 after examining only 30 keyboards. Before 600 keyboards were examined the average number of collisions fell to 5.5. Indeed, the method examined 160 keyboards to reduce the number of collisions by 90%. The power of the scheme is remarkable considering the fact that the problem is inherently NP-hard and the number of distinct keyboards in the solution space is combinatorially explosive.

To compare our scheme with the evolutionary method described in Section 8.3, in Table 8.1 we have also reported the average number of collisions in the dictionary as a function of the number of keyboards that are examined by the optimized version of Levine and Minneman's solution [52, 53, 65]. In this case the average initial number of collisions is reported as 65.5. Note that an exact value for this figure cannot be obtained in the case of evolutionary algorithms because the latter require an initial starting population of keyboards as opposed to a single initial keyboard required by the KLA. Thus the average initial value which we have reported is exactly the same as we have in the case of the KLA. A starting population of 15 keyboards were used to trigger the evolutionary process. The average number of collisions fell to 40 after examining 30 keyboards, and this diminished only to a terminal value of 23 when the experiment was terminated after 600 keyboards were examined. The method examined 390 keyboards to reduce the number of collisions by 90% and even at this juncture the average number of collisions was as high as 27.5.

Apart from reporting the initial and the final values of $\Delta(H, \Pi)$ we have also plotted the average variation of $\Delta(H, \Pi)$ as a function of the number of keyboards examined. This plot is shown in Figure 8.5. The advantage of the KLA over the method due to Levine and Minneman is obvious.

It was observed that the convergence of the KLA could be hastened by constraining every key to have between two and four characters. The effect of imposing such a constraint on the convergence is also reported in Table 8.1. The advantage gained by using this *a priori* information and constraining the number of keys to be between two and four is also obvious.

Finally, to present a "worst case" scenario, the KLA was assigned the task of

learning the optimal keyboard after starting from the worst possible initial keyboard. In this case, all the characters in the alphabet were initially assigned to the same key. Thus, initially, all the words of the same length collided with each other as they all possessed the same representation. From this configuration the KLA processed the words sequentially. The results we have obtained are truly remarkable. The initial number of collisions was 1067. The average number of collisions fell to the amazing low value of 21.8 after examining 60 keyboards. The average final number of collisions was 5.7 when the experiments were terminated, i.e., after examining 600 keyboards. The plot of the number of collisions as a function of the number of keyboards examined is given in Figure 8.6. Also in Figure 8.7a we have given a sample initial worst case keyboard in which all the keys are assigned to the digit 5. In Figure 8.7b, the final assignment of the characters which yields 9 collisions is shown. The power of the scheme is obvious.

Algorithm	Mean Initial $\Delta(H, \Pi)$	Mean Final $\Delta(H, \Pi)$	Mean Time To Converge
EVOL-M	65.5	23.0	390
KLA-U	65.5	5.5	160
KLA-R	58.9	4.9	150

Table 8.1: Comparison of the three algorithms used for keyboard optimization. The results were obtained as an average of 10 experiments with random initial keyboards.

The results that we have presented above are typical. Hundreds of other experiments have been conducted using subsets of the above dictionary and by varying the size of the set of keys, K .

Figure 8.5: Rate of Convergence of the three algorithms used for keyboard optimization.

Figure 8.6: Improvement in Collisions with number of models examined. The initial configuration of characters represents the worst case.

1	2	3
4	5 ABCDEFGHIJ KLMNOPQRS TUVWXYZ	6
7	8	9

(a) The initial keyboard. All the letters are assigned to the same key.

1 HM	2 KT	3 GPUY
4 IS	5 JLOQVXZ	6 ABR
7 CN	8 EW	9 DF

(b) The final keyboard. This keyboard yields 9 collisions.

Figure 8.7: This figure illustrates the performance of KLA in the "worst-case" scenario. The initial keyboard and the final keyboard yielding 9 collisions have been shown.

8.6 Summary

In this chapter we have studied the problem of optimizing a keyboard for a particular finite dictionary, H defined as a subset of the words over a finite alphabet, $\mathcal{A}$. The letters of $\mathcal{A}$ are assigned to elements of a set $\mathbf{K}$, which is the set of integers between unity and K . More explicitly, associated with every element $i \in \mathbf{K}$ is a set C_i such that the union of all C_i 's is $\mathcal{A}$ and all pairwise intersections of C_i 's are null. The aim of the optimization problem is to compute the set $\{C_i\}$ so that if every character in $\mathcal{A}$ is replaced by the index of the set in which it belongs, the transformed version of H has the minimum number of collisions. Thus we aim to compute the sets C_i so as to minimize the ambiguity in such a keyboard assignment process. In this chapter we have proved the result that the problem is inherently NP-hard. Subsequently, the only known solution to the problem has been described and a new learning automaton solution to the problem has been proposed. Experimental results demonstrating the power of the automaton have been presented. A summary of these results have appeared in [77].

Chapter 9

Conclusions

The problems studied in this thesis are related to the application of learning algorithms to the areas of data retrieval and storage. This chapter summarizes the results presented in this thesis and suggests the nature of extensions that could be attempted. We shall cover the problems in the same order as they have been presented in the thesis.

In Chapters 2 and 3 we have studied the problem of approximating the probability densities for discrete and continuous random vectors. The analytical results obtained in Chapter 2 pertain only to vectors whose features are binary valued. It was shown in Chapter 2 that for vectors with binary-valued components the chi-squared metric I_χ always locally follows the EMIM metric I^* . It would be clearly desirable to investigate if this property holds true in general. Also of interest would be a general characterization of the family of distributions for which I_χ always yields the optimal dependence tree. In connection with continuous random variables, the equivalence of I_χ and I^* for distributions other than the normal distribution can be investigated.

In Chapter 4 we have studied the problem of distinguishing between several sources generating random strings. The source recognition is based on first capturing the characteristics of the source in a suitable parameter vector which is used subsequently to achieve the recognition. The problem has been studied only from the point of view of supervised learning. In other words, in our approach, the im-

plixit assumption is that the number of sources as well as the type of strings being emitted (i.e., arbitrary/permutations) by the sources was known to the classifier. It would be interesting to consider the problem when one (or both) of these restrictions is (are) removed. It appears that it will be easier to tackle the situation in which the number of sources is the only unknown. In this scenario one would merely estimate the relevant parameter vector (i.e., either the S or the U vector) and attempt an unsupervised clustering in the parameter space so as to obtain an estimate of the number of classes. The approach to be taken when the type of strings is far from trivial.

In Chapter 5 we have determined the *sufficient* conditions to be satisfied by distributions for which the SWAP heuristic for Doubly Linked Lists (DLLs) yields a lower expected cost than the Move-to-End (MTE) heuristic. The sufficient condition was determined by examining the probabilities of precedence between all pairs of records. The problem of proving that SWAP is superior to MTE under all distributions is open. It is clear that a general proof of this conjecture must take into consideration the explicit expression for the expected cost for the SWAP and MTE heuristics. This approach is likely to be analytically extremely difficult. Another possible avenue to tackle the problem would be to seek for a mapping between the DLL scenario and a SLL scenario. In other words, we would like to be able to achieve the effect of a MTE or SWAP in the DLL context, by applying the appropriate heuristic in the SLL domain, and transforming the hypothetical SLL to yield the resulting DLL. For concreteness, let us denote the list obtained by accessing the record R using the DLL heuristic H , when the DLL is in the configuration π as $H(\pi, R)$. We are interested in finding a mapping f and a heuristic H' in the SLL domain such that $H(\pi, R) = f^{-1}(H'(f(\pi), R'))$. Since the superiority of Transposition (TR) over Move-to-Front (MTF) in the SLL domain is known, such a mapping may help settle the question for DLLs.

In Chapter 5 we have also suggested an absorbing DLL heuristic and derived the necessary conditions under which it converges to the optimal order with a probability of unity. We note that it does not “work” for distributions other than the family

we have identified. It would be interesting to devise an absorbing scheme which is asymptotically optimal for all access distributions. Even the existence of such a scheme is open.

In Chapter 6 we have proposed two new models for a nonstationary query generation. These two models are the Markovian Switching Query Generator (MSQG) and Random Query Generator (RQG). The MTF heuristic has been analyzed under these models of query generation. The analysis of the MTF heuristic is simplified due the fact that the chain (the composite chain in the case of MSQG) for the MTF heuristic is lumpable. The states in the lumped chain directly reflect the information about one record preceding another record. This way we have avoided the necessity of dealing with a chain with an exponential number of states. A similar analysis of the TR heuristic has not been embarked on. It would be interesting to pursue the study of the TR heuristic when interacting with nonstationary query generator.

Chapter 7 is another attempt at enlarging the domain of queries so as to permit queries to include any of the subsets of the data structure being maintained. In this chapter we have proposed a model for query generation called the Multiple Access Query Generator (MAQG) which permits such queries. Variants of the well known SLL heuristics MTF and TR have been suggested, and these can be used in the environment presenting these generalized queries. These generalizations of MTF and TR have been analyzed. In particular, we have shown them to be superior to the heuristics which just operate on the basis of the sequence of records being requested, and which ignore the information about the set boundaries. A natural extension to the work reported here is to investigate data reorganizing heuristics for these generalized queries in the context of other data structures such as the DLL, the Binary Search Tress, Heaps etc.

In the last chapter, namely Chapter 8 we have considered the problem of allocating the symbols of a finite alphabet to the keys of a keyboard in which the number of keys is less than the cardinality of the alphabet. The aim in this case is to attain an assignment of symbols which encodes a finite dictionary with the least possible ambiguity. The minimization problem on hand was shown to be NP-hard. Sub-

sequently we have presented a learning-automaton based solution to this problem. The keyboard optimization problem could be studied under a different model for the dictionary which utilizes the n-gram statistics of the language from which the dictionary is derived. It would be interesting to devise solutions to the keyboard optimization problem based on Variable Structure Stochastic Automata (VSSA).

Also it would be desirable to develop a good understanding of the similarities and differences between the solutions to the Object Partitioning Problem (OPP) and the Keyboard optimization problem. This improved understanding could lead the way to automata based (approximate) solutions to other NP-complete or NP-hard problems.

Bibliography

- [1] A.V. Aho, D.S. Hirschberg, and J.R. Ullman, "Bounds on the Complexity of the Longest Common Sub-Sequence Problem", *J. ACM*, Vol. 23, 1976, pp. 1-12.
- [2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. 1974.
- [3] B. Allen, and J.I. Munro, "Self-Organizing Binary Search Trees", *J. ACM*, Vol. 25, 1978, pp. 526-535.
- [4] A.E. Anderson, P. Nash, and R.R. Weber, "A Counterexample to a Conjecture on Optimal List Ordering", *J. Appl. Prob.*, Vol. 19, 1982, pp. 730-732.
- [5] D.M. Arnow, and A.M. Tenenbaum, "An investigation of the Move-ahead-k Rules", *Proc. of the Thirteenth Southeastern Conf. on Combinatorics, Graph Theory and Computing*, Florida, Feb. 1982, pp. 47-65.
- [6] J.L. Bentley, and C.C. McGeoch, "Amortized Analyses of Self-organizing Sequential Search Techniques", *Comm. ACM*, Vol. 28, 1985, pp. 404-411.
- [7] J.R. Bitner, "Heuristics that Dynamically Organize Data Structures", *SIAM J. Computing*, Vol. 8, No. 1, 1979, pp. 82-110.
- [8] P. Brückner, "On the Complexity of Clustering Problems", in R. Henn, B. Korte and W. Oletti (eds.), *Optimierung und Operations Research*, Lecture notes in Economics and Mathematical Systems, Springer-Verlag, Berlin, 1978.
- [9] P.J. Burville, and J.F.C. Kingman, "On a Model for Storage and Search", *J. Appl. Prob.*, Vol. 10, 1973, pp. 697-701.

- [10] R.P. Cheetham, B.J. Oommen, and D.T.H. Ng, "Adaptive Structuring Of Binary Search Trees Using Conditional Rotations", Tech. Rep. SCS-TR-126, School Of Computer Science, Carleton University, K1S 5B6.
- [11] C.K. Chow and C.N. Liu, "Approximating Discrete Probability Distributions Using Dependence Trees", *IEEE Trans. Inform. Theory*, Vol. 14, 1968, pp. 462-467.
- [12] C.K. Chow, S.S.M. Wang, and, J.H. Siegel, "Sequential Classification of Patient Recovery Patterns After Coronary Artery Bypass Graft Surgery", *Computers and Biomedical Research*, Vol. 12, 1979, pp. 589-613.
- [13] F.R.K. Chung, D.J. Hajela and P.D. Seymour, "Self-Organizing Sequential Search and Hilbert's Inequalities", *J. Comp. and Systems Sciences*, Vol. 36, 1988, pp. 148-157.
- [14] C.R. Cook and D.J. Kim, "Best Sorting Algorithms for Nearly Sorted Lists", *Comm. ACM*, Vol. 23, No. 11, 1980, pp. 620-624.
- [15] K. De Jong, "Adaptive System Design: A Genetic Approach", *IEEE Trans. Syst. Man Cybern.*, 1980, pp. 566-574.
- [16] D.E. Denning, *Encryption and Data Security*. Addison-Wesley, Reading, Mass., 1983.
- [17] G. Dewey, *Relative Frequency of English Speech Sounds*, Harvard Univ. Press. Cambridge, Mass., 1923.
- [18] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.
- [19] R.J. Enbody and H.C. Du, "Dynamic Hashing Schemes", *ACM Computing Surveys*, Vol. 20, No. 2, 1988, pp. 85-113.

- [20] R. Fagin, J. Nevergelt, N. Pippenger, and H.R. Strong, "Extendible Hashing – A Fast Access Method for Dynamic Files", *ACM Trans. Database Systems*, Vol. 4, No. 3, 1979, pp. 315-344.
- [21] G.D. Forney, "The Viterbi Algorithm", *Proc. IEEE*, Vol. 61, 1973.
- [22] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, London, 1972.
- [23] M.R.Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.
- [24] M.R.Garey, D.S. Johnson, and L. Stockmeyer, "Some Simplified NP-Complete Graph Problems", *Theoretical Computer Science*, Vol. 1, pp. 237-267.
- [25] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass., 1989.
- [26] G.H. Gonnet, J.I. Munro, and H. Suwanda, "The Exegesis of Self-Organizing Linear Search", *SIAM J. Computing*, Vol. 10, No. 1, 1981, pp. 613-637.
- [27] P.A.V. Hall and G.R. Dowling, "Approximate String Matching", *ACM Computing Surveys*, Vol. 12, 1980, pp. 381-402.
- [28] M. Hammar, and A. Chan, "Index Selection in a Self-adaptive Database Management System", *Proc. of the ACM SIGMOD Conf.*, 1976, pp. 1-8.
- [29] D.J. Harper, and C.J. Van Rijsbergen, "An Evaluation of Feedback in Document Retrieval Using Co-Occurrence Data", *J. Documentation*, Vol. 34, No. 3, 1978, pp. 189-216.
- [30] K.N. Hendricks, "The Stationary Distribution of an Interesting Markov Chain", *J. Appl. Prob.*, Vol. 9, No. 1, 1972, pp. 231-233.
- [31] K.N. Hendricks, "An Extension of a Theorem Concerning an Interesting Markov Chain", *J. Appl. Prob.*, Vol. 10, No. 4, 1973, pp. 886-890.

- [32] K.N. Hendricks, "An Account of Self-Organizing Systems", *SIAM J. Computing*, Vol. 5, 1976, pp. 715-723.
- [33] J.H. Hester, D.S. Hirschberg, "Self-Organizing Linear Search", *ACM Computing Surveys*, Vol. 17, No. 3, 1985, pp. 295-311.
- [34] D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences", *Comm. ACM*, Vol. 18, 1975, pp. 341-343.
- [35] J.W. Hunt, and T.G. Szymanski, "A Fast Algorithm for Computing Longest Common Subsequences", *Comm. ACM*, 1977, pp. 350-353.
- [36] D.L. Isaacson and R.W. Madsen, *Markov Chains: Theory and Applications*, John Wiley, New York, 1976.
- [37] S. Karlin, and H.M. Taylor, *A First Course in Stochastic Processes*, Academic Press, New York, 1975, 2nd Edition.
- [38] R.L. Kashyap and B.J. Oommen, "Spelling Correction Using Probabilistic Methods", *Pattern Recognition Letters*, Vol. 2, 1984, pp. 147-154.
- [39] R.L. Kashyap, and B.J. Oommen, "An Effective Algorithm for String Correction using Generalized Edit Distances -I. Description of the Algorithm and its Optimality", *Information Sciences*, Vol. 23, 1981, pp. 123-142.
- [40] J.G. Kemeny, and J.L. Snell, *Finite Markov Chains*, Van Nostrand, New York, 1960.
- [41] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, 1983, pp. 671-680.
- [42] G.D. Knott, "Expandable Open Addressing Hash Table Storage and Retrieval", *Proc. ACM SIGFIDET Workshop of Data Desc., Access, and Control*, 1971, pp. 186-206.

- [43] D.E. Knuth, *The Art Of Computer Programming: Vol.1 Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1981.
- [44] D.E. Knuth, *The Art Of Computer Programming: Vol.2 Seminumerical Algorithms*, Addison-Wesley, Reading, Mass., 1981.
- [45] D.E. Knuth, *The Art Of Computer Programming. Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.
- [46] H.H. Ku and S. Kullback, "Approximating Discrete Probability Distributions", *IEEE Trans. Inform. Theory*, Vol. 15, No. 4, 1968, pp. 444-447.
- [47] S. Lakshmivarahan, *Learning Algorithms: Theory and Applications*, Springer-Verlag, New York, 1981.
- [48] K. Lam, M. Leung, and M.K. Siu, "Self-Organizing Files with Dependent Accesses", *J. Appl. Prob.*, Vol. 21, 1984, pp. 343-359.
- [49] P. Larson, "Dynamic Hashing". *BIT*, Vol. 18, 1978, pp. 184-201.
- [50] R. Lawrence and R.A. Wagner, "An Extension of the String to String Correction Problem", *J. ACM*, Vol. 22, 1975, pp. 177-183.
- [51] C. Levcopoulos and O. Petersson, "Heapsort — Adapted for Presorted Files", *Proc. of the WADS Workshop, Ottawa, Canada, August 1989*, Springer-Verlag, pp. 499-509.
- [52] S.H. Levine, "An Adaptive Approach to Optimal Keyboard Design for NonVocal Communication", *Proc. Intl. Conf. Cybernetics and Society*, 1985, pp. 334-337.
- [53] S.H. Levine, S.L. Minneman, C.O. Getschow, and C. Goodenough-Trepagnier, "Computer Disambiguation Of Multi-Character Text Entry: An Adaptive Design Approach", *Proc. IEEE Intl. Conf. on Systems, Man Cybern.*, 1986, pp. 298-301.

- [54] W. Litwin, "Virtual Hashing: A Dynamically Changing Hashing", *Proc. Very Large Databases Conf.*, Berlin, 1978, pp. 517-523.
- [55] S.Y. Lu, and K.S. Fu, "A Sentence-to-Sentence Clustering Procedure for Pattern Analysis", *Proc. of the IEEE COMPSAC Conf.*, 1977, pp. 492-498.
- [56] J.R. Magnus, and H. Neudecker, *Matrix Differential Calculus: with Applications in Statistics and Econometrics*, John Wiley, New York, 1988.
- [57] D. Maier, "The Complexity of Some Problems on Subsequences and Supersequences", *J. ACM*, Vol. 25, 1978, pp. 322-336.
- [58] H. Mannila, "Measures of Presortedness and Optimal Sorting Algorithms", *IEEE Trans. Computer*, Vol. 34, No. 4, 1985, pp. 318-325.
- [59] W.J. Masek, and M.S. Paterson, "A Faster Algorithm Computing String Edit Distances", *J. Comp. and Systems Sciences*, Vol. 20, 1980. pp. 18-31.
- [60] D. Matthews, D. Rotem, and E. Bretholz, "Self-Organizing Doubly-Linked Lists", *J. Comput. Maths.*, Sec.A, Vol. 8, 1980, pp. 99-106.
- [61] K. Mehlhorn, "Sorting Presorted Files", *Proc. of the Fourth GI Conf. on Theoretical Comp. Sc.*, Springer-Verlag, 1979, pp. 199-212.
- [62] K. Mehlhorn, *Data Structures and Algorithms : Vol. 1: Sorting And Searching*, Springer-Verlag, Berlin/Heidelberg, F.R. Germany, 1984.
- [63] R.S. Michalski, "A Theory and Methodology of Inductive Learning", in *Machine Learning: An Artificial Intelligence Approach.*, Ed. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell.
- [64] S.L. Minneman, "A Simplified Touch-Tone Telecommunication Aid for Deaf and Hearing Impaired Individuals", *Proc. Eighth Annual RESNA Conf.*, Memphis, 1985, pp. 209-211.

- [65] S.L. Minneman, "KeyBoard Optimization Technique to Improve the Output Rate Of Disabled Individuals", *Proc. Ninth Annual RESNA Conf.*, Minneapolis, 1986, pp. 402-404.
- [66] K.S. Narendra and M.A.L. Thathachar, *Learning Automata: An Introduction*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [67] K.S. Narendra, and M.A.L. Thathachar, "Learning Automata - A Survey", *IEEE Trans. Syst. Man Cybern.*, Vol. 4, No. 4, 1974, pp. 323-334.
- [68] D.L. Neuhoff, "The Viterbi Algorithm as an Aid in Text Recognition", *IEEE Trans. Inform. Theory*, Vol. 21, 1975, pp. 222-226.
- [69] D.T.H. Ng, and B.J. Oommen, "Generalizing Singly-Linked Reorganizing Heuristics For Doubly-Linked Lists", *Proc. of 1989 Conf. on Mathematical Foundations of Comp. Sc.*, Rytro, Poland, Aug-Sept. 1989, pp. 380-389.
- [70] B.J. Oommen, and D.C.Y. Ma, "Deterministic Learning Automata Solutions to the Equi-Partitioning Problem", *IEEE Trans. Computer*, Vol. 37, No. 1, 1988, pp. 2-14.
- [71] B.J. Oommen and D.C.Y. Ma, "Fast Object Partitioning Using Stochastic Learning Automata". *Proc. of the 1987 Intl. Conf. on Research and Development in Information Retrieval*, New Orleans, 1987, pp. 111-122.
- [72] B.J. Oommen, and E.R. Hansen, "List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations", *SIAM J. Computing*, Vol. 16, No. 4, 1987, pp. 705-716.
- [73] B.J. Oommen, E.R. Hansen, and J.I. Munro, "Deterministic Move-to-Rear List Organizing Strategies with Optimal and Expedient Properties", To appear in *Theoretical Computer Science*.
- [74] B.J. Oommen, and D.T.H. Ng, "Optimal Constant Space Move-to-Rear List Organization", *Proc. of the 1989 Intl. Symp. on Optimal Algorithms*, Varna, Bulgaria, May-June 1989.

- [75] B.J. Oommen and D.T.H. Ng, "Arbitrarily Distributed Random Permutation Generation", *Proc. of 1989 ACM Comp. Sci. Conf.*, Louisville, KY, Feb 1989, pp. 27-32. Also available as Tech. Rep. SCS-TR-138 from the School Of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6.
- [76] B.J. Oommen, and D.T.H. Ng, "Ideal List Organization for Stationary Environments", Tech. Rep. SCS-TR-154, School Of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6.
- [77] B.J. Oommen, R.S. Valiveti, and J.R. Zgierski, "An Adaptive Learning Solution to the Keyboard Optimization", submitted for publication. A summary of these results have appeared in *Proc. of the Third Intl. Conf on Industrial Engineering Applications of AI and Expert Systems*, IEA/AIE 90, Charleston, SC, July 1990. pp. 981-990.
- [78] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan-Kaufmann, San Mateo, 1988.
- [79] K. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes : The Art of Scientific Computing*, Cambridge University Press, New York, 1986.
- [80] S.S. Rao, *Optimization Methods: Theory and Applications*, John Wiley, New York, 1980.
- [81] R.L. Rivest, "On Self-Organizing Sequential Search Heuristics", *Comm. ACM*. Vol. 19, No. 2, 1976, pp. 63-67.
- [82] S.M. Ross, *Introduction to Probability Models* Academic Press, New York, 1980, 2nd Edition.
- [83] D. Sankoff, and J.B. Kruskal, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Mass., 1983.

- [84] R. Shinghal, and G.T. Toussaint, "Experiments in Text Recognition with the Modified Viterbi Algorithm", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 1, 1979, pp. 184-192.
- [85] S. Srihari, *Computer Text Recognition and Error Correction*, IEEE Computer Society Press, 1984.
- [86] M. Tamminen, "Extendible Hashing with Overflow", *Information Processing Letters*, Vol. 15, No. 5, 1982, pp. 227-232.
- [87] A.M. Tenenbaum, and R.M. Nemes, "Two Spectra of Self-Organizing Sequential Search Algorithms", *SIAM J. Computing*, Vol. 11, No. 3, 1982, pp. 557-566.
- [88] J.P. Tremblay and P.G. Sorenson, *An Introduction to Data Structures with Applications*, McGraw-Hill, New York, 1976.
- [89] M.L. Tsetlin, *Automaton Theory and the Modelling of Biological Systems*, Academic Press, New York and London, 1973.
- [90] M.L. Tsetlin, "On the Behavior of Finite Automata in Random Media", *Automation and Control*, Vol. 22, 1962, pp. 1210-1219. Originally appeared in *Automatika Telemekhanika*, Vol. 22, 1961, pp. 1345-1354.
- [91] R.S. Valiveti and B.J. Oommen, "A New Metric for Determining Dependence Trees for Pattern Recognition", *Proc. of the 1989 Intl. Symp. on Computer Arch. and Digital Signal Processing*, Oct. 1989, Hong Kong, pp. 474-479.
- [92] R.S. Valiveti and B.J. Oommen, "The Optimality of the Chi-squared Metric For Determining Dependence in Normally Distributed Vectors", *Proc. of Intl. Symp. on Information Theory and its Applications, ISITA '90*, Nov. 1990, Hawai, pp. 375-378.
- [93] R.S. Valiveti, and B.J. Oommen, "Recognizing Sources of Random Strings", to appear in *IEEE Trans. Pattern Anal. Machine Intell.*. A summary of these

- results appear in *Proc. Twenty-fourth Conf. on Inform. Systems. and Sciences*, Princeton, NJ, March 21-23, 1990, pp. 601-610.
- [94] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm", *IEEE Trans. Inform. Theory*, Vol. 13, 1967, pp. 260-269.
- [95] C.J. Van Rijsbergen, "A Theoretical Basis For the Use of Co-occurrence Data in Information Retrieval", *J. Documentation*, Vol. 33, No. 2, 1977, pp. 106-119.
- [96] C.J. Van Rijsbergen, *Information Retrieval*, Butterworths, London, 1979.
- [97] R.A. Wagner and M.J. Fisher, "The String-to-String Correction Problem", *J. ACM*, Vol. 21, 1974, pp. 168-173.
- [98] S.K.M. Wong, and F.C.S. Poon, "Comments on Approximating Discrete Probability Distribution with Dependence Trees", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 11, No. 3, 1989, pp. 333-335.
- [99] A.K.C. Wong, and D.K.Y. Chiu, "Synthesizing Statistical Knowledge from Incomplete Mixed-Mode Data", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 9, No. 6, 1987, pp. 796-805.
- [100] A.K.C. Wong, and T.S. Liu, "Typicality, Diversity, and Feature Pattern of an Ensemble", *IEEE Trans. Computer*, Vol. 24, No. 2, 1975, pp. 158-181.
- [101] C.K. Wong and A.K. Chandra, "Bounds on the String Editing Problem", *J. ACM*, Vol. 23, 1976, pp. 13-16.
- [102] C.T. Yu, M.K. Siu, K. Lam, and F. Tai, "Adaptive Clustering Schemes: General Framework", *Proc. of the IEEE COMPSAC Conf.*, 1981, pp. 81-89.
- [103] C.T. Yu, C. Suen, K. Lam, and M.K. Siu, "Adaptive Record Clustering", *ACM Trans. Database Systems*, Vol. 10, 1985, pp. 180-204.

Appendix A

Proof of Theorem 2.1

In Section 2.2, it was mentioned that the maximum likelihood estimate of the dependence tree can be obtained by **Algorithm Chow**. The result is indeed powerful and not obvious at first sight. This part of the paper is an attempt to expand on the (outline of) the proof that appeared in [11]. To simplify the details we will be proving the theorem only for the case of binary valued features, but the generalization will become obvious, once the reader follows this proof.

Proof:

Let $X^1, X^2, \dots, X^s$ be the s *independent* observations (given the underlying dependence tree t). The likelihood function is then:

$$L_t(X^1, X^2, \dots, X^s) = \prod_{k=1}^s P_t(X^k)$$

where $P_t(X^k)$ is the joint probability that the vector X^k is derived from the particular tree t . If we use the product rule for the distribution generated by this tree (2.3), we get the following equation for the Likelihood function:

$$L_t(X^1, X^2, \dots, X^s) = \prod_{k=1}^s \prod_{i=1}^N Pr(x_{m_i}^k | x_{m_i(t)}^k)$$

Instead of maximizing $L_t(X^1, X^2, \dots, X^s)$, we attempt to maximize its logarithm, and denote it by $l_t(X^1, X^2, \dots, X^s)$. Therefore,

$$l_t(X^1, X^2, \dots, X^s) = \log L_t(X^1, X^2, \dots, X^s)$$

$$\begin{aligned}
&= \sum_{k=1}^s \sum_{i=1}^N \log \{Pr(x_{m_i}^k | x_{m_{j(i)}}^k)\} \\
&= \sum_{i=1}^N \sum_{k=1}^s \log \{Pr(x_{m_i}^k | x_{m_{j(i)}}^k)\} \tag{A.1}
\end{aligned}$$

the final expression being obtained by interchanging summations.

The maximum likelihood tree is the value $t = \tau$, for which the above sum is maximized. Now consider the two features x_{m_i} and $x_{m_{j(i)}}$. From the definition of the dependence tree, we realize that in the dependence tree t , there is an edge between these nodes. In order to compute the likelihood function as given in (A.1), we need to know the parameters $Pr(x_{m_i} = b_1 | x_{m_{j(i)}} = b_2)$ for $b_1 \in \text{Domain}(x_{m_i})$ and $b_2 \in \text{Domain}(x_{m_{j(i)}})$. If all the features of interest are binary valued, $b_1, b_2 \in \{0, 1\}$. In this case, essentially, we have two independent parameters, these being $Pr(x_{m_i} = 0 | x_{m_{j(i)}} = 0)$ and $Pr(x_{m_i} = 0 | x_{m_{j(i)}} = 1)$. Conceptually, these parameters can be thought as being associated with a branch of the dependence tree. For the $(N-1)$ branches of the tree, we have $2(N-1)$ parameters all of which can be independently chosen.

The reader will now realize that in essence, we are faced with two decisions. The first one involves selecting the best values for the above mentioned parameters -- two for each edge of the tree. The second problem concerns choosing the best estimate τ from the set of all spanning trees. To obtain the Maximum likelihood estimate, we should study the problem of maximizing the sum in (A.1), *for a particular tree*. That is, given a *particular* tree, we are trying to find the best estimates for the conditional probabilities such that the sum is maximized. Note that in (A.1), the term $\sum_{k=1}^s \log \{Pr(x_{m_i}^k | x_{m_{j(i)}}^k)\}$ corresponds to adding the logarithms of the parameters associated with the branch $(x_{m_i}, x_{m_{j(i)}})$ over all the samples. Since the parameters for the individual branches are independent, we can maximize the terms corresponding to each branch and then add the corresponding maximums that have been obtained. That is, for a given tree t ,

$$\max_l l_t(X^1, X^2, \dots, X^s) = \max \left\{ \sum_{i=1}^N \sum_{k=1}^s \log \{Pr(x_{m_i}^k | x_{m_{j(i)}}^k)\} \right\}$$

$$= \sum_{i=1}^N \left\{ \max \sum_{k=1}^s \log \{ Pr(x_{m_i}^k | x_{m_{j(i)}}^k) \} \right\}$$

For simplicity, let $\alpha = m_i$ and $\beta = m_{j(i)}$. The above sum is composed of terms of the form $Pr(x_\alpha = b1 | x_\beta = b2)$, where $b1, b2 \in \{0, 1\}$. Therefore,

$$\begin{aligned} \sigma &\triangleq \sum_{k=1}^s \log \{ Pr(x_{m_i}^k | x_{m_{j(i)}}^k) \} \\ &= \sum_{k=1}^s \log Pr(x_\alpha^k | x_\beta^k) \\ &= \sum_{i,j=0,1} n_{ij} \log Pr(x_\alpha = i | x_\beta = j) \end{aligned}$$

where, n_{ij} is the number of samples in which $x_\alpha = i$ and $x_\beta = j$. Also note that $\sum n_{ij}$ is equal to s , the total number of samples.

Now, if we introduce the notation that, $\delta_{ij} = Pr(x_\alpha = i | x_\beta = j)$, we get,

$$\sigma = n_{00} \log \delta_{00} + n_{01} \log \delta_{01} + n_{10} \log \delta_{10} + n_{11} \log \delta_{11} \quad (A.2)$$

Our aim is to maximize σ under the constraints $\delta_{00} + \delta_{10} = 1$ and $\delta_{01} + \delta_{11} = 1$; the latter conditions being a consequence of the conditional distributions. Simplifying (A.2), subject to the above mentioned constraints, we have,

$$\begin{aligned} \sigma &= \log \left\{ \prod_{i,j} \delta_{ij}^{n_{ij}} \right\} \\ &= \log \left[\underbrace{\{\delta_{00}^{n_{00}} (1 - \delta_{00})^{n_{10}}\}}_{\text{}} \underbrace{\{\delta_{01}^{n_{01}} (1 - \delta_{01})^{n_{11}}\}}_{\text{}} \right] \end{aligned}$$

To maximize σ , we must individually maximize the terms within the square brackets. By using the methods of elementary calculus, We can easily establish that $\{\delta_{00}^{n_{00}} (1 - \delta_{00})^{n_{10}}\}$ and $\{\delta_{01}^{n_{01}} (1 - \delta_{01})^{n_{11}}\}$ attain their maximum values when

$$\begin{aligned} \delta_{00} &= \frac{n_{00}}{n_{00} + n_{10}} \\ \delta_{01} &= \frac{n_{01}}{n_{01} + n_{11}} \end{aligned}$$

respectively. Therefore σ is maximized when the individual probabilities are chosen to be their own Maximum likelihood estimates which are indeed their intuitive

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

frequency based estimates. Now, from (A.1) we have,

$$\begin{aligned}
 l_t(X^1, X^2, \dots, X^s) &= \sum_{i=1}^N \sum_{k=1}^s \log\{Pr(x_{m_i}^k | x_{m_{j(i)}}^k)\} \\
 &= \sum_{i=1}^N \sum_{k=1}^s \log\left\{\frac{Pr(x_{m_i}^k, x_{m_{j(i)}}^k)}{Pr(x_{m_i}^k)Pr(x_{m_{j(i)}}^k)}\right\} + \sum_{i=1}^N \sum_{k=1}^s \log(Pr(x_{m_i}^k)) \\
 &= s \sum_{i=1}^N \hat{I}(x_{m_i}, x_{m_{j(i)}}) + K.
 \end{aligned}$$

where K is independent of the tree but is only a function of the samples. Hence to maximize $l_t(X^1, X^2, \dots, X^s)$, over all possible trees, we only need to compute the MST of the graph with $\hat{I}(x_i, x_j)$ as edge weights. Hence the result.

□

Appendix B

Generating Samples of Normally Distributed Vectors

In this section, we consider the problem of generating a random vector $\mathbf{X}$ whose distribution is $N(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ and Σ are known parameters. The generation technique is based on the following property (stated without proof).

Theorem B.1 *Let $\mathbf{X}$ be the vector $[x_1, x_2, \dots, x_N]^T$ which has the distribution $N(\boldsymbol{\mu}, \Sigma)$. Then the random vector $\mathbf{Y} = A\mathbf{X}$ has the distribution $N(A\boldsymbol{\mu}, A\Sigma A^T)$.*

Corollary B.1 *Let $\boldsymbol{\mu}, \Sigma$ be the mean vector and the covariance matrix of a normal distribution respectively. Let Z be a normal random vector which has the distribution $N(0, I)$. Also let $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_N)$ and Φ be the diagonal eigenvalue matrix, and the eigenvector matrix of Σ respectively. Then*

$$\mathbf{X} = \boldsymbol{\mu} + \Phi\Lambda^{1/2}Z$$

has the normal distribution $N(\boldsymbol{\mu}, \Sigma)$.

Proof:

It is clear that the random vector $\mathbf{X}$ has the mean vector $\boldsymbol{\mu}$. Furthermore, using Theorem B.1, the covariance matrix of $\mathbf{X}$ can be evaluated as:

$$\text{Covariance}(\mathbf{X}) = \Phi\Lambda^{1/2}\text{Covariance}(Z)\{\Phi\Lambda^{1/2}\}^T = \Phi\Lambda^{1/2}I\Lambda^{1/2}\Phi^T = \Phi\Lambda\Phi^T.$$

Because of the fact that Φ is the orthogonal eigenvector matrix for the covariance matrix Σ , it follows that $\Phi\Lambda\Phi^T = \Sigma$. Hence the result. $\square$

The above corollary leads to a straightforward technique to generate samples drawn from a normal distribution, given its parameters. This is given below in Algorithm GenerateNormalSamples (in Program B.1).

Algorithm GenerateNormalSamples(μ, Σ, X)**Input:**

The parameters μ and Σ for the normal distribution obeyed by X .

Output:

A random vector X drawn from the distribution $N(\mu, \Sigma)$.

Memory requirements:

Z – a vector of dimension $N \times 1$

Φ, Λ – square matrices of dimension $N \times N$.

Assumptions:

Assumes that the function “GetUnivariateNormal”, for generating normal variates with the distribution $N(0, 1)$ is available. One method of realizing this function can be found in [44, pp.117-118]

The procedure “GetEigenValuesAndVectors” is assumed to be present, for determining the eigenvalues and the eigenvectors of a real symmetric matrix. The most common procedure for achieving this computation is the Jacobi method described in [79].

Method:

1. GetEigenValuesAndVectors (Σ, Φ, Λ)
1. for $I := 1$ to N do
     $Z[I] := \text{GetUnivariateNormal}$
3. Compute $X = \mu + \Phi\Lambda^{1/2}Z$.

End Algorithm GenerateNormalSamples

Program B.1: Algorithm to Generate Samples from a Multi-variate Normal Distribution.

Appendix C

Computing Closeness Measure Between Two Normal Distributions

In this section, we consider the problem of computing the closeness measure between two normal distributions described as $N(\boldsymbol{\mu}, \Sigma)$ and $N(\boldsymbol{\mu}_a, \Sigma_a)$. The expression for the closeness measure between these distributions was reported in (3.24) and is given by Theorem C.1. The derivation makes use of a central result, which is given as Lemma C.1 below.

Lemma C.1 *Let $\mathbf{X}$ be a random (column) vector of dimension $N \times 1$, whose mean vector is $\boldsymbol{\mu}$ and the covariance matrix is Σ . Then, if A is any square matrix of dimension $N \times N$, the expected value of $\mathbf{X}^T A \mathbf{X}$ is given by:*

$$E[\mathbf{X}^T A \mathbf{X}] = \text{tr}(A\Sigma) + \boldsymbol{\mu}^T A \boldsymbol{\mu}.$$

where $\text{tr}(A)$ denotes the trace of the square matrix A .

Proof:

The proof can be found in [56, pp.247]. □

Using Lemma C.1, we prove the main result of this Appendix.

Theorem C.1 *Let $N(\boldsymbol{\mu}, \Sigma)$ and $N(\boldsymbol{\mu}_a, \Sigma_a)$ denote two normal distributions. Then $I(P, P_a)$, the information theoretic closeness measure between these distributions is given by:*

$$I(P, P_a) = 0.5 \log \frac{|\Sigma_a|}{|\Sigma|} - 0.5 \left\{ N - \text{tr}(\Sigma_a^{-1} \Sigma) - (\boldsymbol{\mu} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\boldsymbol{\mu} - \boldsymbol{\mu}_a) \right\}. \quad (\text{C.1})$$

Proof:

We begin by writing the explicit forms of the the density functions for the two normal distributions under consideration:

$$\begin{aligned} P(\mathbf{X}) &= (2\pi)^{-N/2} |\Sigma|^{-0.5} \exp \left\{ -\frac{1}{2} (\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \right\}, \\ P_a(\mathbf{X}) &= (2\pi)^{-N/2} |\Sigma_a|^{-0.5} \exp \left\{ -\frac{1}{2} (\mathbf{X} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{X} - \boldsymbol{\mu}_a) \right\}. \end{aligned} \quad (\text{C.2})$$

From (C.2) we can easily obtain the ratio of $P(\mathbf{X})$ to $P_a(\mathbf{X})$ as:

$$\begin{aligned} \frac{P(\mathbf{X})}{P_a(\mathbf{X})} &= \frac{|\Sigma_a|^{0.5}}{|\Sigma|^{0.5}} \cdot \exp \left\{ -\frac{1}{2} (\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \right. \\ &\quad \left. + \frac{1}{2} (\mathbf{X} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{X} - \boldsymbol{\mu}_a) \right\}. \end{aligned}$$

or equivalently,

$$\begin{aligned} \log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} &= 0.5 \log \frac{|\Sigma_a|}{|\Sigma|} - \frac{1}{2} (\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \\ &\quad + \frac{1}{2} (\mathbf{X} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{X} - \boldsymbol{\mu}_a). \end{aligned} \quad (\text{C.3})$$

From the definition of the closeness measure $I(P, P_a)$, it clear that this quantity is simply the expected value of the random variable $\log \frac{P(\mathbf{X})}{P_a(\mathbf{X})}$. Thus,

$$\begin{aligned} I(P, P_a) &= E \left[\log \frac{P(\mathbf{X})}{P_a(\mathbf{X})} \right] \\ &= 0.5 \log \frac{|\Sigma_a|}{|\Sigma|} - \frac{1}{2} E \left[(\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \right] \\ &\quad + \frac{1}{2} E \left[(\mathbf{X} - \boldsymbol{\mu}_a)^T \Sigma_a^{-1} (\mathbf{X} - \boldsymbol{\mu}_a) \right] \end{aligned} \quad (\text{C.4})$$

We can make use of Lemma C.1 to compute the expected values of the two quadratic forms that are encountered in (C.4). We first concentrate on computing the expected value of the first quadratic form on the RHS of (C.3). We note that since the mean of the random vector $\mathbf{X}$ is $\boldsymbol{\mu}$, the mean of the random vector $\mathbf{X} - \boldsymbol{\mu}$ is 0. Thus by Lemma C.1:

$$E \left[(\mathbf{X} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{X} - \boldsymbol{\mu}) \right] = \text{tr}(\Sigma^{-1} \Sigma) = N. \quad (\text{C.5})$$

Also noting that $\mathbf{X} - \boldsymbol{\mu}_a$ has the mean $\boldsymbol{\mu} - \boldsymbol{\mu}_a$, a second application of Lemma C.1 yields:

$$E \left[(\mathbf{X} - \boldsymbol{\mu}_a)^T \boldsymbol{\Sigma}_a^{-1} (\mathbf{X} - \boldsymbol{\mu}_a) \right] = \text{tr}(\boldsymbol{\Sigma}_a^{-1} \boldsymbol{\Sigma}) + (\boldsymbol{\mu} - \boldsymbol{\mu}_a)^T \boldsymbol{\Sigma}_a^{-1} (\boldsymbol{\mu} - \boldsymbol{\mu}_a). \quad (\text{C.6})$$

Substituting the results of (C.5) and (C.6) into (C.4) we obtain the result stated in (C.1).

□

END

28.02.92

FIN