INFORMATION TO USERS

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

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

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

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

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Advances in Data Compression and Pattern Recognition

by

Luis Gabriel Rueda
B.C.S., M.C.S.

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

Ottawa-Carleton Institute for Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
April, 2002

© Copyright 2002, Luis Gabriel Rueda

National Library of Canada
Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services
Acquisitions et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

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

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

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

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

0-612-71947-2

Canada

The undersigned recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis
Advances in Data Compression and Pattern Recognition
submitted by
Luis Gabriel Rueda
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Director, School of Computer Science

Supervisor

External Examiner (Ph.D.)

Carleton University, April 2002

Dedicated to my beautiful wife, Ivanna,

and

our beloved daughter, Maria Dolores.

Abstract

In this thesis, we present our contributions to two important areas of *computer science*, namely *data compression* and *pattern recognition.*

In the area of data compression, we introduce an enhanced version of the static Fano coding, based on the concept of the rearrangement of two lists. The algorithm is formally analyzed and rigorously tested empirically. Its superiority over other similar algorithms has been demonstrated.

We introduce a greedy algorithm for the adaptive Fano coding. We formalize the partitioning algorithms for the multi-symbol input and the binary output. The empirical results on the well-known benchmarks demonstrate the advantageous properties of our greedy algorithm. This greedy algorithm has also been extended for the multi-symbol output alphabet.

We also present a more efficient algorithm which uses a new tree structure derived from the binary search tree, namely the Fano binary search tree. This new scheme, the corresponding tree-based operators, and the conditional shifting heuristic used to consistently maintain this Fano tree, have been theoretically analyzed and tested on well-known benchmarks files.

In the area of pattern recognition, we present the formal theory for optimal pairwise linear classifiers, and a formal analysis of why heuristic functions work. With regard to the theory of optimal pairwise linear classifiers, we derive the necessary and sufficient conditions for the classifier to be optimal and as a pair of straight lines for the two-dimensional case, when the underlying distribution is normal. This, in particular, resolves Minsky's paradox, which has been open since 1957. The corresponding classifiers have been empirically analyzed on synthetic and real-life data showing its superiority over the traditional Fisher's approach. These results have also been extended

for the case of multi-dimensional features.

We also present a formal analysis that relates the accuracy of heuristic functions to the optimality of the solutions that heuristic algorithms yield. We prove that given a heuristic algorithm, which could utilize either of two heuristic functions, the more accurate heuristic has a higher probability of yielding a superior solution. This result has also been tested for the database query optimization problem.

Acknowledgments

First of all, I wish to record my most sincere gratitude to Professor B. John Oommen for being one of the best Professors I have ever known. He has provided years of invaluable support to my research while allowing me the freedom to make this thesis what it is. Apart from being an excellent supervisor, John has also been the best advisor I could have had during my stay at Carleton.

John has being a friend, a counselor, and an excellent teacher. He has been beside me at every stage of my studies at Carleton. With his help, I have been able to publish papers in various highly-respected journals and conference proceedings. He has also helped me in improving my analytical skills when proving our theoretical results. Thanks, John, for your kindness – God bless you!

I want to thank Professor Anil Maheshwari for his advise in the preparation of my thesis proposal and in my search for a job. Many thanks to Professor Eric Dubois for his constructive comments on canonical codes. My sincere gratitude to Professor Kaizhong Zhang for his constructive remarks which helped in improving the quality of this thesis. I would also like to express my thanks to Professor Nelson Rodriguez and Professor Sergio Zapata for all their assistance when I first moved to Canada.

I want to thank the following friends and colleagues who provided me with technical assistance and friendship during my student days: Gerardo Reynaga who gave me all the technical assistance that I needed, even on his busiest days; Miguel Vargas for his true friendship and unconditional support; and Peter Taillon who tirelessly helped me in improving my linguistic skills. Of course, I cannot forget my other colleagues: Abdy, Juan Pablo, Kenji (I hope you forgive the constant knocks on the door from "Amigo") and my numerous other "soccer buddies".

I would like to express my heartfelt gratitude to my parents for providing me with

a good education from the earliest days in life, and for teaching me to work hard and realize the value of that which is good.

Many thanks to the School of Computer Science at Carleton University for helping me financially and providing the conducive atmosphere in which to work. In particular, I thank Sandy Dare, Nicole Enouy, Linda Pfeiffer, and Marlene Wilson. I am grateful to the "Departamento de Informática", National University of San Juan, Argentina, for providing me with financial support for my graduate work. I am grateful to the National Sciences and Engineering Research Council of Canada for its financial support during the last part of my thesis.

In closing, I am indebted to my wife Ivanna. This thesis would never have been completed if it were not for her standing beside me, through "thick and thin". I dedicate this thesis to her and our lovely daughter, Maria Dolores.

Contents

I Thesis Overview **1**

1 Introduction **2**

1.1 Data Compression 3

1.1.1 Overview 3

1.1.2 Static Compression 6

1.1.3 Adaptive Compression 6

1.2 Research Problems in Data Compression 8

1.2.1 The Enhanced Fano Coding 8

1.2.2 Adaptive Fano's Method 10

1.2.3 Adaptive Fano Coding using BSTs 13

1.3 Pattern Recognition 15

1.4 Research Problems in Pattern Recognition 17

1.4.1 Linear Classifiers 17

1.4.2 Pairwise Linear Classifiers and Minsky's Paradox 19

1.4.3 Heuristic Functions: Accuracy and Optimality 21

1.5 Outline of the Thesis 23

II Data Compression: Overview and Research **28**

2 Data Compression **29**

2.1 Coding Theory 30

2.1.1 An Encoding System 30

2.1.2 Sources 30
2.1.3 Codes 33
2.2 Information Theory 36
2.2.1 Introduction 36
2.2.2 Information 36
2.2.3 Entropy 38
2.2.4 Higher-order Sources 40
2.3 Compression Measures 44
2.4 Static Coding 47
2.4.1 Huffman's Algorithm 48
2.4.2 Shannon's Method 52
2.4.3 Fano's Method 54
2.4.4 Arithmetic Coding 56
2.4.5 Run Length Coding 59
2.5 Adaptive Coding 60
2.5.1 Huffman Coding 61
2.5.2 Fano's Method 63
2.5.3 Shannon's Method 63
2.5.4 Arithmetic Coding 63
2.5.5 Compression and Adaptation 65
2.5.6 Other Adaptive Methods 67
2.5.6.1 Interval Coding 67
2.5.6.2 Recency Rank Coding 67
2.5.6.3 Predictive Coding 69
2.6 Higher-order Coding 70
2.6.1 Huffman Coding 71
2.6.2 Dictionary Based Coding 72
2.6.2.1 The LZ77 Compression Scheme 72
2.6.2.2 The LZ78 Compression Scheme 74
2.6.3 Dynamic Markov Compression 75
2.6.4 Prediction with Partial Matching 78

2.6.5 Block Sorting Compression 79
2.6.6 Grammar-Based Coding 80
2.6.7 Other Higher-order Methods 81
2.7 Conclusions 82

3 Enhanced Fano Coding Schemes 83
3.1 The Enhanced Coding Algorithms 85
3.1.1 Motivation 85
3.1.2 The Enhanced Coding Algorithms 89
3.2 Properties of the Enhanced Fano Coding 90
3.3 Empirical Results 98
3.4 Conclusions 101

4 Greedy Adaptive Fano Coding 102
4.1 The Encoding Algorithm 103
4.2 The Decoding Algorithm 106
4.3 Correctness of the Greedy Adaptive Fano Coding 108
4.4 Complexity of the Greedy Fano Algorithms 114
4.5 Empirical Results 115
4.6 Greedy Adaptive Fano Coding for Multi-symbol Code Alphabets 117
4.6.1 Nearly-Equal-Probability Partitioning 117
4.6.2 Sub-Optimal Nearly-Equal-Probability Partitioning 125
4.7 Conclusions 128

5 Adaptive Fano Coding Using Binary Search Trees 130
5.1 Binary Search Trees 131
5.1.1 Adaptive Structuring of BSTs 132
5.2 The Use of Conditional Rotations in BSTs 133
5.2.1 Rotation Operations 133
5.2.2 Conditional Rotation Heuristics 135
5.3 Adaptive Fano Coding Using BSTs 137
5.3.1 Motivation 137

5.3.2 Fano BSTs 138
5.3.3 Shifting Operations in Partitioning BSTs 145
5.3.3.1 The Shift-To-Left Operator 146
5.3.3.2 The Shift-To-Right Operator 158
5.3.4 The Conditional Shifting Heuristic 169
5.3.5 The Encoding Algorithm 172
5.3.6 The Decoding Algorithm 174
5.3.7 Properties of the Fano BST Coding 176
5.4 Empirical Analysis 185
5.4.1 Speed and Compression Efficiency 185
5.4.2 Variability of the Fano BST 188
5.5 Graphical analysis 190
5.6 Conclusions 194

III Pattern Recognition: Overview and Research 199

6 Pattern Recognition 200
6.1 Statistical Pattern Recognition 201
6.1.1 Random Vectors 201
6.1.2 Diagonalization 203
6.1.2.1 Simultaneous Diagonalization 204
6.2 Pattern Classification 206
6.2.1 Bayes Decision Theory 206
6.3 Pairwise Linear Classifiers 208
6.3.1 Single Hyperplane Classifiers 208
6.3.2 Optimal Pairwise Linear Classifiers 210
6.4 Heuristic Functions: Accuracy and Optimality 211
6.5 Conclusions 213

7 2-Dimensional Optimal Pairwise Linear Classifiers 215
7.1 Pairwise Linear Classifiers : Cases 216

7.1.1 Diagonalized Classes 216
7.1.2 Different Covariances and Different Means 224
7.1.3 Different Covariances and Equal Means 228
7.2 Pairwise Classification 229
7.2.1 The Discriminant Function 229
7.2.2 Pairwise Linear Classification 231
7.3 Simulation Results on Synthetic Data 234
7.4 Pairwise Linear Classifiers in Real Life Data 239
7.4.1 Parameter Approximation 239
7.4.2 Empirical Results 242
7.5 Conclusions 245

8 *d*-Dimensional Optimal Pairwise Linear Classifiers 246
8.1 Multi-Dimensional Pairwise Classifiers for Diagonalized Classes 247
8.2 Linear Discriminants with Different Means 251
8.3 Linear Discriminants with Equal Means 253
8.4 Simulation Results 254
8.4.1 Linear Discriminants for Two Diagonalized Classes 254
8.4.2 Pairwise Linear Discriminant with Different Means 255
8.4.3 Pairwise Linear Discriminant with Equal Means 256
8.4.4 Analysis of Accuracy 258
8.5 Conclusions 258

9 A Formal Analysis of Why Heuristic Functions Work 260
9.1 Problem Motivation 265
9.1.1 Applications 265
9.1.2 Problem Statement 267
9.1.3 Restrictions of Our Model 268
9.2 Heuristic Functions : Accuracy vs. Optimality 269
9.2.1 Analysis Using Exponential Distributions 270
9.2.2 Analysis Considering Normal Distributions 277
9.3 Simulation Results for Database Heuristic Functions 283

9.3.1 Empirical Results 283
9.3.2 Geometric Justification of the Rationale 284
9.4 Conclusions 287

IV Contributions and Future Directions 288

10 Conclusions 289
10.1 Enhanced Static Fano Coding 290
10.1.1 Contributions 290
10.1.2 Future Directions 290
10.2 Greedy Adaptive Fano Coding 291
10.2.1 Contributions 291
10.2.2 Future Directions 292
10.3 Adaptive Fano Coding using Binary Search Trees 293
10.3.1 Contributions 293
10.3.2 Future Directions 295
10.4 Optimal Pairwise Linear Classifiers 295
10.4.1 Contributions 295
10.4.2 Future Directions 296
10.5 A Formal Analysis of Why Heuristic Functions Work 297
10.5.1 Contributions 297
10.5.2 Future Directions 297

List of Figures

1.1 The process of generating the encoding scheme for Fano's method, where the source symbols are $\{a, b, c, d\}$, with probabilities of occurrence being $[.4, .3, .2, .1]$. 9

1.2 Generation of the encoding scheme using Fano's method for the source symbols $\{a, b, c, d\}$ whose frequency counters are $[4, 3, 2, 2]$. 11

1.3 Code word generation process for the source symbol 'c' by following the new partitioning strategy of the Fano coding. Note that the symbols given by '?', and the corresponding segments of the tree, need not be processed. 12

1.4 The Fano tree constructed by following the principles of the Fano coding, where the source alphabet is $\{a, b, c, d\}$ and the frequency counters are $[4, 3, 2, 2]$. 13

1.5 The updated Fano tree after 'c' is encoded. The updated frequency counters are $[4, 3, 3, 2]$. The counters and indices in bold are those of the updated nodes. 14

1.6 Optimal linear classifier between two normally distributed classes. The covariance matrices are equal. 18

1.7 A simple scenario for the 2-bit binary-featured classes. Minsky showed that no single linear classifier can be found for these classes. 20

2.1 Data Encoding and Decoding. 30

2.2 Huffman tree constructed with $\mathcal{S} = \{a, b, c, d\}$ with $\mathcal{P} = [.4, .3, .2, .1]$, and $\mathcal{A} = \{0, 1\}$. 52

2.3 Fano encoding of the source $\mathcal{S} = \{a, b, c, d\}$ with frequencies $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$. 55
2.4 Predictive encoding and decoding. 69
2.5 Huffman tree for the context a, constructed with $\mathcal{S} = \{a, b, c, d\}$, $\mathcal{A} = \{0, 1\}$, and $\mathcal{P} = [.5, .25, .125, .125]$ extracted from $\mathcal{P}^{(1)}$ given in Example 2.5. 72
2.6 Example of an FSM representing an instance of the statistical information of a higher-order source using DMC. 76
2.7 Finite State Machine representing the statistical information of the higher-order source given in Example 2.14. 77
2.8 The Grammar-Based Coding transforming and encoding. 81

3.1 An example of a coding tree that does not satisfy the sibling property. 87
3.2 Two different coding trees constructed with the source alphabet and probabilities of occurrence given in Example 3.2. 94

5.1 A rotation operation performed on node t_i. 134
5.2 An example of a Fano BST constructed from $\mathcal{S} = \{a, b, c, d, e\}$ and $\mathcal{P} = [8, 3, 3, 3, 3]$. 141
5.3 Partitioning BST construction procedure for $\mathcal{S} = \{a, b, c, d, e\}$, $\mathcal{P} = [8, 3, 3, 3, 3]$, and $\mathcal{A} = \{0, 1\}$. The resulting tree is a Fano BST. 142
5.4 An example showing an STL (Case STL-1) operation performed on a partitioning BST, $\mathcal{T}$, constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$. The resulting partitioning BST, $\mathcal{T}'$, is the one of (b). The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 149

5.5 In (a), a partitioning BST is constructed from the source alphabet $\mathcal{S} = \{a, b, c, d, e\}$ whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3]$. In (b), the partitioning BST is obtained after performing an STL operation (Case STL-2) on node t_i. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 153

5.6 A partitioning BST, $\mathcal{T}$, constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$, and the corresponding partitioning BST, $\mathcal{T}'$, after performing an STL-3 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 155

5.7 A partitioning BST constructed from $\mathcal{S} = \{a, b, c, d, e, f\}$ and $\mathcal{P} = [3, 3, 3, 3, 2, 1]$. In (a), the partitioning BST before performing the STR-1 operation, and in (b), the resulting partitioning BST. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 160

5.8 A partitioning BST constructed from the source alphabet $\mathcal{S} = \{a, b, c, d\}$ and $\mathcal{P} = [2, 2, 2, 1]$, and the corresponding tree after performing an STR-2 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 164

5.9 Two partitioning BSTs. The tree on the top was constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [3, 1, 1, 1, 1, 1, 1]$, and includes the changes to be done when applying an STR-3 operation on t_i. The tree on the bottom is the resulting partitioning BST after the STR-3 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself. 166

5.10 Plot of the percentage of encoding steps with at least one shift operation for the files of the Calgary corpus. 192

5.11 Percentage of encoding steps with shift operations vs. number of encoding steps for the files of the Canterbury corpus. 192

5.12 Number of shift operations per node visited in the entire encoding process for the files of the Calgary corpus. 193
5.13 Plot of the number of shift operations per node visited vs. size of the output (in bits) for the files of the Canterbury corpus. 194
5.14 Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *book1* which was obtained from the Calgary corpus. 195
5.15 Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *paper1* which was obtained from the Calgary corpus. 195
5.16 Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *cp.html* which was obtained from the Canterbury corpus. . . 196
5.17 Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *kennedy.xls* which was obtained from the Canterbury corpus. 197

6.1 Plot of the probability density function for the two-dimensional normally distributed random vector of Example 6.1. 202
6.2 Simultaneous diagonalization: orthonormal and whitening transformations. 205
6.3 Optimal linear classifier between two normally distributed classes, ω_1 and ω_2, given in Example 6.3. 209
6.4 Two overlapping classes, ω_1 and ω_2, with equal means, and possible ways of classification using straight lines. 210

7.1 Example of pairwise linear discriminant in diagonalization for the case described in Theorem 7.2. The data set is DD-1. 235
7.2 Example of pairwise linear discriminant in diagonalization for the case described in Theorem 7.2. The data set is DD-2. 235
7.3 Example of pairwise linear discriminant with different means for the case described in Theorem 7.6. The data set is DM-1. 236
7.4 Example of pairwise linear discriminant with different means for the case described in Theorem 7.6. The data set is DM-2. 237
7.5 Example of pairwise linear discriminant with equal means for the case described in Theorem 7.7. The data set is EM-1. 238

7.6 Example of pairwise linear discriminant with equal means for the case described in Theorem 7.7. The data set is EM-2. 238
7.7 Optimal quadratic classifier, Fisher's discriminant function, and the approximated pairwise linear classifier used to classify the "area" and "smoothness" features from the WDBC data set. 244

8.1 Example of pairwise linear discriminant for diagonalized normally distributed classes. This example corresponds to the data set DD-1. . . . 255
8.2 Example of pairwise linear discriminant with different means for the case described in Section 8.2. These classes corresponds to the data set DM-1.256
8.3 Example of pairwise linear discriminant with equal means for the case described in Section 8.3. The data set is EM-1. This resolves the generalized multi-dimensional Minsky's paradox. 257

9.1 An example of doubly exponential distributions for the random variables $X_1^{(opt)}$, $X_2^{(opt)}$, $X_1^{(subopt)}$ and $X_2^{(subopt)}$, whose parameters are $\lambda_1 = 0.4$ and $\lambda_2 = 0.2$. 271
9.2 Function $G(\alpha_1, k)$ plotted in the ranges $0 \leq \alpha_1 \leq 1$ and $0 \leq k \leq 1$. . . . 277
9.3 An example showing the probability density function of four normal random variables whose parameters are $\sigma_1 = 1$, $\sigma_2 = 1.4$, $\mu_1 = 0$, and $\mu_2 = 4$. 279
9.4 Function $G(\sigma_1, k\sigma_1)$ plotted in the ranges $1 \leq \sigma_1 \leq 10$ and $1 \leq k\sigma_1 \leq 10$, where $\sigma_2 = k\sigma_1$. 282
9.5 Estimated probability density function for three doubly exponential random variables that represent the error in estimation for the Equi-width, the Equi-depth and the R-ACM. 286
9.6 Expected values for a doubly exponential random variable, and the actual costs obtained after optimizing queries on 400 random databases using the R-ACM histogram. 286

List of Tables

2.1 Arithmetic coding of the source word $\mathcal{X} = adbc$, where the source alphabet is $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet is $\mathcal{A} = \{0, 1\}$. 59

3.1 A brief description of the files of the Calgary Corpus used to test the efficiency of the compression algorithms discussed in this research work. 98

3.2 Description of the files of the Canterbury corpus used as a benchmark for the compression algorithms presented in this research work. 99

3.3 Empirical results obtained after compressing test files from the Calgary corpus by using the static Huffman's algorithm, the static Fano's method, and the sibling-enhanced version of the static Fano's method (Fano$^+$). 100

3.4 Details of the experiments performed with the static version of Huffman's algorithm, the traditional static Fano's method, and the sibling-enhanced version of static Fano's method on files of the Canterbury corpus. 100

4.1 An empirical comparison of the greedy adaptive version of Fano's method and the adaptive Huffman coding which were tested on files of the Calgary corpus. 116

4.2 Empirical Results obtained after running the adaptive Huffman coding and greedy adaptive Fano's method on files of the Canterbury corpus. . 116

5.1 Speed and compression ratio for the Fano BST coding and the greedy adaptive Fano method tested on files of the Calgary corpus. 186
5.2 Speed and compression ratio obtained after running the greedy adaptive Fano coding and the Fano BST coding on files of the Canterbury corpus. 187
5.3 Empirical results for the variability of the Fano BST obtained after running the adaptive Fano coding using BSTs on files of the Calgary corpus. 188
5.4 Variability of the Fano BST during the encoding process on the files of Canterbury corpus. 189

7.1 Accuracy in classification of test points generated with the six examples presented above. The accuracy is given in percentage of points correctly classified. 239
7.2 Accuracy in classification obtained from the Fisher's discriminant approach, and the approximated optimal pairwise linear classifier on the WDBC data set. 243

8.1 Accuracy of classification of 100 three dimensional random test points generated with the parameters of the examples presented above. The accuracy is given in percentage of points correctly classified. 258

9.1 Ratio between the probability of making the wrong decision for two normally distributed random variables whose standard deviations are σ_1 and σ_2. 281
9.2 Simulation results for the R-ACM, the Equi-width, and the Equi-depth, after optimizing queries on 400 randomly generated databases. The column labeled "R > W" contains the number of times in which R-ACM obtained a better solution than the Equi-width on 100 randomly generated databases. The information contained in the other columns has a similar interpretation, where "R", "W" and "D" stand for the R-ACM, the Equi-width and the Equi-depth respectively. The last row contains the sum of the values in each column. 284

List of Algorithms

1 **Static_Huffman_Encoding** 50
2 **Static_Huffman_Decoding** 51
3 **Static_Fano_Encoding** 55
4 **Static_Arithmetic_Encoding** 57
5 **Adaptive_Huffman_Encoding** 62
6 **Adaptive_Arithmetic_Encoding** 64
7 **Enhanced_Fano_Encoding** 91
8 **Enhanced_Fano_Decoding** 92
9 **Greedy_Adaptive_Fano_Encoding** 105
10 **Greedy_Adaptive_Fano_Decoding** 107
11 **All_Consecutive_Partitionings** 122
12 **Greedy_r-ary_Adaptive_Fano_Partitioning** 126
13 **Fano_BST_Construction** 140
14 **STL_Operation** 157
15 **STR_Operation** 168
16 **Fano_BST_Updating** 171
17 **Fano_BST_Encoding** 173
18 **Fano_BST_Decoding** 175

Part I

Thesis Overview

Chapter 1

Introduction

The advent of the digital computer has changed the very fabric of society causing a technological revolution that has been growing in the last five decades. The power and myriad applications of computers have presented us with massive volumes of information which are difficult to manage efficiently even with the most powerful hardware. The information to be managed involves disciplines so varied as medicine, business, engineering, mathematics, and the social sciences.

The expansion of this technological revolution has resulted in the creation of several new well-defined areas, including networking, parallel computation, neural networks, evolutionary computation, pattern recognition, data compression, cryptography, databases, real time systems, and software engineering. Although these areas have different applications, they share common practical requirements: efficient storage and acquisition, processing, and transmission of information.

Most of these areas are based on formal concepts derived from other scientific disciplines such as mathematics and physics, while others have matured since a couple of millennia. For example, *data compression* dates back from the history of Roman civilization, in which a form of shorthand writing was devised to record the Emperor's speeches. In spite of this, the foundational theory of data compression was proposed in the late 1940's, pre-dating the invention of the first digital computer. Another example is the area of *pattern recognition* (PR), a prominent field of *machine intelligence*, which itself is a well-developed field. Numerous books have been written about the theory and

applications of PR, which include face and speech recognition, fingerprint identification, character recognition, medical diagnosis, etc. To highlight how established this field is, it suffices to mention that one of the most important methods of determining linear classifiers, Fisher's approach, was proposed in the late 1930's.

As this research thesis focuses on two of these areas, data compression and pattern recognition, we shall first present a *brief* overview of these areas in the following sections.

This thesis is composed of four parts. The first part contains the introduction to the fields of study and the problems addressed in this thesis. The second part contains an overview of lossless data compression, and discusses all our contributions to this field. The third part includes a brief introduction to the field of pattern recognition, the formal theory of pairwise linear classifiers, and the use of PR techniques to derive a formal analysis of why heuristics work. The fourth part discuss the conclusions of the thesis.

1.1 Data Compression

1.1.1 Overview

The area of data compression evolved from very ancient times – records of this "art" have been found from relics obtained from the era of the Roman civilization. As mentioned previously, a form of shorthand writing was invented to record the Emperor's speeches, which typically, had to be written down at the same time of the speech. Little work was done in data compression until the nineteenth century, when Braille developed a reading system for the blind. The basis of this powerful system is applied even today in efficient compression methods. The main idea in data compression is to assign shorter symbolic representations to the most frequent symbols, such as the vowel 'e'. Conversely, symbols that are less likely to occur, such as the letter 'z', are assigned longer representations.

It can be said that data compression is the art of representing a stream of data by a, hopefully, shorter one. Whether or not the original data can be completely

recovered from the compressed data defines the compression scheme to be *lossless* or *lossy*, respectively. Our research focuses on lossless data compression, the field which we briefly discuss below. A more complete and detailed survey of the field is found in Chapter 2.

The fundamental theory of data compression is based on the field of information theory. This field was formalized by C. Shannon in 1948 [147], and was intended to be a mathematical basis of communications. Apart from data compression, the foundations of information theory have also been used in other areas of computer science, including cryptography, machine learning, etc. The most important contribution of Shannon's theory was to mathematically measure the amount of information contained in a symbol, and the introduction of the concept called "entropy" of a source.

Any source has an associated alphabet, or set of symbols, whose elements have some probabilities of occurrence. Given a code alphabet, the idea of compressing a stream of data is to replace each source symbol in the input by a code word, which is a sequence of one or more code alphabet symbols. As mentioned above, shorter code words are intended to be assigned to the most frequently occurring symbols. As a consequence of this, one expects a reduction in the average (or expected) code word length source symbol, in the output. The following example will help to clarify this issue.

Example 1.1. Suppose that the source alphabet is composed of four symbols, a, b, c and d whose probabilities of occurrence are 0.5, 0.3, 0.15 and 0.05, respectively, and the code alphabet is the binary alphabet. If an uncompressed representation of $a \rightarrow 00$, $b \rightarrow 01$, $c \rightarrow 10$, $d \rightarrow 11$ is used, two bits are required per input symbol. A more efficient assignment is $a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, and $d \rightarrow 111$. Observe that the shorter code word, 0, is assigned to the most likely symbol, a, and that the average code word length per symbol is 1.7, which is less than two. □

Note that the code generated in Example 1.1, satisfies the *prefix condition*. Under this property, no code word can be a prefix of any other code word. A prefix code ensures lossless compression.

Shannon showed that given a source alphabet and associated probabilities, a stream of data can be losslessly compressed up to a lower bound, called the *average information*

amount or *entropy*. From this fundamental theorem, he proposed a coding method which assigns a code word to each symbol, where the length of the code word is proportional to the amount of information contained.

Although Shannon's method is straightforward and has a fast execution time, it is far from efficient with regard to the compression ratio. The most efficient coding method, in the sense that it minimizes the average code word length, was proposed by D. Huffman in 1952 as a project in a course taught by a famous mathematician R. Fano at M.I.T. In 1961, Fano proposed a less efficient method (with respect to the compression ratio), but the latter was faster than Huffman's algorithm. Various problems associated with these methods, mainly when dealing with the binary source alphabet, led researchers to propose the family of *arithmetic coding* schemes. Although most of the implementation techniques involving arithmetic coding have been developed recently, its basic philosophy came from the ideas of C. Shannon and P. Elias.

Observe that in Example 1.1, we considered the probabilities of the symbols occurring independently of each other. This is what is known as the *zeroth-order* model for the source. On the contrary, *higher-order* models consider the probabilities of the symbols given the occurrence of other symbols. For example, in English text, the occurrence of an '*h*' after a '*t*' is very likely. Also, the occurrence of an '*e*' after '*th*' is even more likely, not to mention the occurrence of a blank space after word '*the*'. The fundamental theory of higher-order models (also known as Markov models) was proposed by A. Markov, a famous mathematician of the nineteenth century. Markov models have successfully been used in other areas of computer science as well.

It is important to note that Shannon's result can lead to even more compression if a higher-order source model is used. Generally speaking, the higher the order, the more the compression attainable. In spite of this, there is a serious drawback when increasing the order of the source model. In a kth-order model, m^{k+1} probabilities are to be stored, and hence m^k encoding schemes have to be constructed and stored, where m is the number of source symbols. For example, if the source alphabet is the ASCII code set, one needs to store and process 256^2 probabilities for a first order model. In practice, a good Markov model is typically a sixth-order model, for which $256^7 \approx 10^{19}$ probabilities are to be stored !

In order to overcome this situation, many encoding methods have been developed, which cleverly store only part of the m^{k+1} probabilities or the m^k different encoding schemes. One of the most widely used methods are the LZ family of algorithms, also known as *dictionary-based* methods, proposed by J. Ziv and A. Lempel in 1977 and 1978. Other models, which use even less resources and obtain better compression results, have also been developed, including dynamic Markov compression (DMC), prediction with partial matching (PPM), the Burrows-Wheeler transform (BWT), grammar-based coding (GBC), etc. These methods are discussed in more detail in Chapter 2.

1.1.2 Static Compression

In Example 1.1, we constructed an encoding scheme which consists of assigning 0 to a, 10 to b, and 11 to c. This was done by assuming that the probabilities of these symbols are known, e.g. 0.5, 0.3, and 0.2. Suppose that we want to encode the source sequence *aabac*. The corresponding output sequence is 0010011, obtained by replacing each source symbol by its corresponding code word. Note that the same encoding scheme and probabilities of occurrence were used in encoding the *entire* source sequence. This type of encoding is known as *static coding* or *static compression*.

Static coding is important in applications that do not require on-line data processing, such as in databases, historical data storage, backups, image compression, etc. In addition, algorithms that allow extremely fast decoding have been developed for a particular family of static codes, namely the *canonical codes*. These codes are discussed in Chapter 3. Another important application of static coding is for sources in which the statistical information about the source is known *a priori*. The encoding process can thus be done in a "single pass" without the necessity of learning the statistical distribution of the source.

1.1.3 Adaptive Compression

Adaptive coding (or *adaptive compression*) works by simultaneously encoding and learning about the probability distribution of the source symbols. As a result of this, the

corresponding code words will also be changing during the encoding process. These changes are based on a *learning* mechanism that considers the history of the symbols already encoded, and updates their probabilities. The simplest and most widely used learning mechanism in adaptive compression consists of assigning a *frequency counter* to each symbol, which is initialized to an arbitrary value, e.g. unity. As a new symbol appears in the input sequence, its frequency counter is incremented by unity. The probability of each symbol is then approximated by dividing its frequency counter by the sum of all the frequency counters.

The other phase of adaptive compression consists of, at each step, updating the encoding scheme used. Historically, the first proposed method, whose adaptive version was presented by N. Faller in 1973 [45], is Huffman's algorithm. Given a code alphabet with r symbols, the static version of this method operates by creating a particular r-ary tree, namely the *Huffman tree.* This particular tree, whose construction method and properties are discussed latter in Chapter 2, is then used to generate the encoding scheme that encodes the entire source sequence.

The adaptive version of Huffman's algorithm consists of keeping an on-line Huffman tree, which itself is updated at each step of the encoding process. When a new symbol arrives, it is encoded using the current tree, and a new tree is created by updating the relevant segments of the old tree using the new (updated) probability distribution of the source symbols. The literature reports two independent approaches presented by R. Gallager [48] and D. Knuth [75] respectively. They both generate the code word of the input symbol using the *current* Huffman tree, and generate a *new* Huffman tree, which is, in turn, used in the next encoding step.

Using similar learning mechanisms, but with completely different structures, an adaptive version of arithmetic coding has also been proposed. The procedure and some examples of this version can be found in [58].

The adaptive versions of the encoding methods have some advantages over the corresponding static versions. Adaptive methods do not require *a priori* knowledge of the symbol probabilities, which are, later, updated during the encoding process. In contrast, static methods assume that these probabilities are available prior to encoding. A common solution to this problem is to do a first pass over the input data to "learn"

the symbol probabilities, and a second pass to encode the data. However, there are numerous applications, such as those in the area of communications, including the internet, ftp, audio and video streaming, instant messaging, network broadcasting and multicasting, etc., in which static encoding is not feasible.

To the best of our knowledge, no adaptive version of Fano's method has been proposed in the literature. This is one of the problems that we address in this research work, and which we discuss later in Section 1.2.2. The latter constitutes a fairly greedy, simplistic approach to the adaptive version of Fano coding. We also present a much more efficient approach that utilizes Binary Search Trees (BSTs), introduced in Section 1.2.3, and presented in detail in Chapter 5.

1.2 Research Problems in Data Compression

1.2.1 The Enhanced Fano Coding

As mentioned earlier, Huffman's algorithm proceeds by generating an r-ary tree, namely the *Huffman tree*. This tree is constructed in a bottom-up fashion. In contrast, the static version of Fano's method creates a Fano tree, but in a top-down fashion. We illustrate how the Fano tree and the encoding scheme are created by the following example.

Example 1.2. Consider the source alphabet symbols, $\{a, b, c, d\}$, whose probabilities are $[.4, .3, .2, .1]$, and the binary code alphabet.

First of all, the symbols have to be arranged in a non-increasing order of probability. The Fano tree is created by partitioning the list of symbols in two[1] sublists whose sums of probabilities are as nearly-equal as possible. We assign '0' to the left branch and '1' to the right branch, and hence, to those symbols in each sublist. We thus have two sublists, $\{a\}$ and $\{b, c, d\}$, where '0' is assigned to a, and '1' to b, c, and d. At the same time, two new nodes are created in the Fano tree, which are the left and right children of the current node (in this case, the root of the tree). We recursively continue in the same fashion for those sublists with more than one symbol, i.e. $\{b, c, d\}$, and obtain

[1] In fact, we divide the set of symbols in r groups, where r is the cardinality of the code alphabet.

symbol	probability	code word	partitioning
a	.4	0	0
b	.3	10	0
c	.2	110	0
d	.1	111	1

Figure 1.1: The process of generating the encoding scheme for Fano's method, where the source symbols are $\{a, b, c, d\}$, with probabilities of occurrence being $[.4, .3, .2, .1]$.

two groups $\{b\}$ and $\{c, d\}$, and so on. The final encoding scheme is: $a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, and $d \rightarrow 111$.

The corresponding partitioning procedure, and the encoding scheme generation process are illustrated in Figure 1.1. □

In more realistic examples (such as the ASCII code set), if we start with the list of source symbols in a non-increasing order of probabilities of occurrence, it can be seen that, in general, symbols with higher probabilities can sometimes be assigned to longer code words than those with lower probabilities. This condition is not desirable, since, as a result, compression is less efficient. In order to rectify this condition, we propose a superior Fano-based scheme in Chapter 3. We introduce a front-end process to the static Fano encoding scheme, which achieves a code word rearrangement in such a way that the code word lengths are sorted in an increasing order of lengths.

The importance of using Fano's method in static coding is based on a particular family of codes, namely *canonical codes*. A canonical code is a code that satisfies the following properties:

(*i*) The code words are arranged in an increasing order of length.

(*ii*) The code words are arranged in a lexicographical order.

For example, if the lengths of the code words are $[2, 2, 3, 3, 3, 3]$, the corresponding code words are to be $\{00, 01, 100, 101, 110, 111\}$.

Canonical codes are very important in data compression because they allow *extremely* fast decoding, and require approximately *half* of the space used by a decoding tree. Obtaining canonical codes using Huffman's algorithm is not straightforward – it requires the execution of very complicated mechanisms involving a three-phase process for calculating the code word lengths, and then invoking an extra process for obtaining the canonical code itself. Using Fano coding, instead, a canonical code is generated in a natural way. This can be observed in Figure 1.1 in which the code words generated are arranged in a lexicographical order.

The enhanced version of the static Fano coding that we present in this research work utilizes a concept which we called "code word rearrangement", and which is based on a fundamental property of two lists (the code words in terms of their lengths and probabilities) arranged in an increasing and decreasing order respectively. These new results will be presented in Chapter 3 of the thesis.

1.2.2 Adaptive Fano's Method

The importance of adaptive methods is primarily due to the fact that they do not need, *a priori* the statistical information about the source to start the encoding process. They proceed with the encoding, and simultaneously update the symbol probabilities. As mentioned in Section 1.1.3, this feature is crucial in communications applications, such as the internet, ftp, audio and video streaming, instant messaging, network broadcasting and multicasting, etc.

The simplicity of Fano's method in building the coding tree, and the fact that Fano's method achieves an average code word length which is very close to the minimal, has motivated us to study how we can devise adaptive versions of the Fano coding process.

Adaptive versions of Huffman's algorithm and arithmetic coding have been proposed in the literature. As part of our research contribution, we present two approaches to adaptive Fano coding. These are a greedy version, which is mentioned below, and a Fano-based coding model that utilizes a special version of the binary search tree (BST), namely the Fano BST.

Consider the source alphabet with the associated probabilities as in Example 1.2.

symbol	freq. counter	code word	partitioning
a	4	00	0
b	3	01	1
c	2	10	0
d	2	11	1

Figure 1.2: Generation of the encoding scheme using Fano's method for the source symbols $\{a, b, c, d\}$ whose frequency counters are $[4, 3, 2, 2]$.

Suppose that we are to encode the source sequence $dcac\ldots$. At the current step in the encoding process, we are to encode the symbol 'd' using the current Fano encoding scheme, i.e. the one depicted in Figure 1.1. This implies that '111' will be sent to the output. After the corresponding code word is sent to the output, the probabilities (or the frequency counters) of the source symbols must be updated so that they efficiently represent the distribution of the source in the next encoding step. Suppose that the frequency counters are maintained for the source symbols, namely $[4, 3, 2, 1]$. After increasing the counter of 'd' by unity, we obtain the counter vector to be $[4, 3, 2, 2]$, which will be used to generate the Fano encoding scheme for the next source symbol, 'c'. This procedure is depicted in Figure 1.2.

Observe that when the first partitioning is done, the list $[4, 3, 2, 2]$ could have been partitioned into either $[4]$ and $[3, 2, 2]$, or $[4, 3]$ and $[2, 2]$, in order to maintain the properties of the Fano coding. In the next encoding step, the source symbol 'c' will be encoded as '10', and not as '110', which would have been the case if the encoding scheme of Figure 1.1 were used. Observe thus that different encoding schemes are being used at different time instants.

One of the drawbacks of this scenario is that we have to re-generate the *entire* encoding scheme at each encoding step, and choose the corresponding code word for the current source symbol. The frequency counters are updated, and if necessary, the list of symbols is rearranged so that it is maintained in a non-increasing order of the frequency counters.

One of the contributions that we present in this thesis, is the *greedy adaptive* version

symbol	freq. counter	code word	partitioning
a	4	?	
b	3	?	
c	2	10	0 1
d	2	?	

Figure 1.3: Code word generation process for the source symbol 'c' by following the new partitioning strategy of the Fano coding. Note that the symbols given by '?', and the corresponding segments of the tree, need not be processed.

of the Fano coding, which avoids the re-generation of the entire encoding scheme at each step. Additionally, this scheme maintains *only* the list of symbols and their probabilities of occurrences (in terms of their frequency counters), both arranged in a non-increasing order of probability of occurrence. At each encoding step, only the code word of the symbol being encoded is generated.

We illustrate this with a single example, and explain the details and formalities later. Suppose that we are to encode the source symbol 'c', and the current list of symbols and their probabilities of occurrence (frequency counters) are $\{a, b, c, d\}$ and $[4, 3, 2, 2]$ respectively. Observe that we could generate the entire encoding scheme (like the one depicted in Figure 1.2), compute the corresponding code word for 'c', and transmit it to the output. However, at the current encoding step, we do not need the code words for a, b, and d. Alternatively, we propose to partition the lists of symbols and frequency counters, and choose the sublist that contains the symbol that we want to encode, which in this case is 'c'. We send '0' or '1' to the output depending on which list 'c' is contained in (in our example, '1'), and proceed with the sublist that contains 'c', namely $\{c, d\}$. We then partition $\{c, d\}$ into $\{c\}$ and $\{d\}$. Since 'c' is contained in the first sublist, $\{c\}$, we send a '0' to the output, and stop the partitioning process because we have a sublist with a single symbol. This process is illustrated in Figure 1.3, in which it can be observed that '10' is the codeword generated and sent to the output, and that the other code words are not generated. The counter of 'c' is then incremented by unity, yielding $[4, 3, 3, 2]$, and hence no rearranging is necessary.

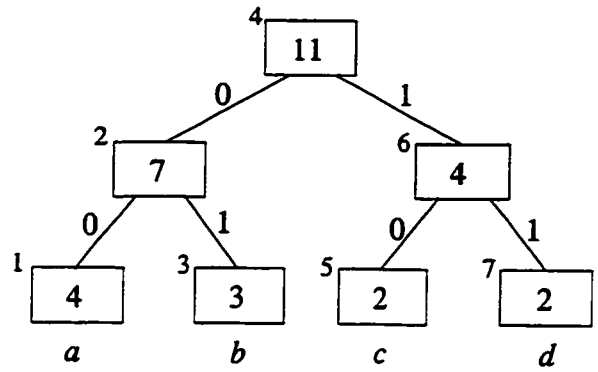


Figure 1.4: The Fano tree constructed by following the principles of the Fano coding, where the source alphabet is $\{a, b, c, d\}$ and the frequency counters are $[4, 3, 2, 2]$.

In Chapter 4, we present the formalities of this solution. The chapter formally presents the encoding and decoding algorithms, their correctness, and the other theoretical properties of this solution.

1.2.3 Adaptive Fano Coding using BSTs

Observe in Figure 1.3 that we can generate the code word for the corresponding source symbol at each encoding step, without having to generate the entire encoding scheme. In fact, to generate a single code word, we have to reconstruct part of the Fano tree. In the first partitioning step, we have to find the index that partitions the list of symbols into two sublists. Thus, it takes $O(\log n)$ time to merely generate a single bit of the corresponding code word. Additionally, the same procedure must be followed to generate the next code word bit, and so on.

Alternatively, we propose to maintain the Fano tree that we use, at each time instant, to encode the current source symbol. Thus, for the source symbols $\{a, b, c, d\}$, whose frequency counters are $[4, 3, 2, 2]$, the corresponding Fano tree, which is constructed by following the principles of the Fano coding, is the one depicted in Figure 1.4.

Observe that each leaf is associated with a source symbol, and each edge is labeled with a '0' for the left branch, and with a '1' for the right branch. Also, a number is

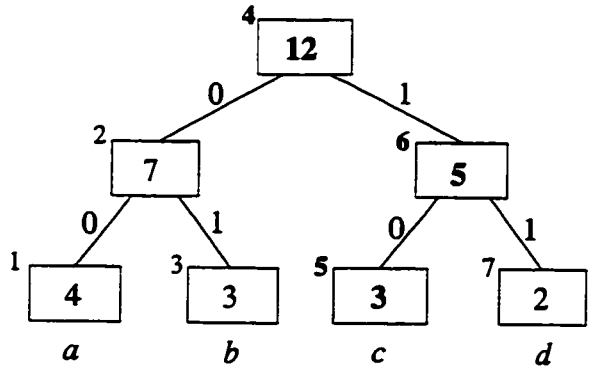


Figure 1.5: The updated Fano tree after 'c' is encoded. The updated frequency counters are $[4, 3, 3, 2]$. The counters and indices in bold are those of the updated nodes.

shown beside the top-left corner of each node, which represents the index of the node in the list of nodes obtained by traversing the tree from left to right. It is clear that the code word for 'c', '10', (or for that matter, any source symbol) can be generated in $O(\log n)$ time. At the same time, the tree must be updated by incrementing the counter of each node visited, namely nodes 4, 6, and 5 (in this order). As a result, a new tree is obtained, which is depicted in Figure 1.5.

Note that the tree is also updated in $O(\log n)$ time. What remains is to ensure that the updated tree is a Fano tree. This is done while the nodes 4, 6 and 5 are updated. Good fortune meets us in this particular example, because the updated tree satisfies the properties of the Fano coding. Although reconstructing the Fano tree may take a long time, it can be seen that as the encoding process proceeds, if the source is stationary, the changes made in the Fano tree converge to zero, and hence, in most cases, a single symbol will be encoded in $O(\log n)$ time.

On the other hand, we argue that the indices of the nodes can be used in a prudent manner. To encode the symbol 'c' using the tree of Figure 1.4, we can perform a *binary search* of node 5. A little insight demonstrates that the nodes of the tree are arranged as in a BST, where the keys are the indices beside the top-left corner of the nodes. We know that 'c' is the third symbol of the list, and thus, $(3)(2) - 1 = 5$ is the corresponding node in the tree. This implies that we can search for node number '5' as if it were searched in a BST. To achieve this, we can start from the root of the tree,

compare '4' and '5', and go to the right. Since $4 < 6$, we go to the left, and locate the corresponding leaf, namely the one associated with 'c'. Note that the code word generation and the tree updating are both done using a binary search process.

Discovering this has motivated us to develop an adaptive version of the Fano coding using BSTs. No similar technique has even been reported. Indeed, this new model utilizes a new kind of BSTs, which we call the *Fano BST*, invoking two new operators, the Shift-To-Left and the Shift-To-Right operators. The Fano BST, the operators, and the formalism for the tree and the corresponding algorithms are the subject matter of Chapter 5.

1.3 Pattern Recognition

The second half of the thesis solves various open problems in Pattern recognition (PR) – a field of machine intelligence that has been one of the dominant technologies in the last few decades. Broadly speaking, PR is a decision-making process, based on some *a priori* knowledge of the domain in which patterns are being recognized. More specifically, the system has some information about the features of a set of classes. Given an object, and this information, the system attempts to characterize an object as belonging to one of the known classes with as high accuracy as possible.

A typical PR system consists of a sensor (a camera or a voltage detector, for example), a feature processing mechanism (an algorithm that makes the system learn about the classes, namely the *training* phase), and a classification mechanism which is an algorithm that classifies a given object as belonging to a class. This phase is called namely the *testing* phase.

During the training phase the system receives the *training set* as the input, which contains samples that belong to the different classes. From these training samples, the system stores the relevant information to be used in the testing phase. During the latter, the *testing set* is received, and the PR system classifies each testing sample with a certain accuracy.

There are many applications of PR including face and speech recognition, fingerprint identification, character recognition, medical diagnosis, etc. In each of these

applications, the information about the classes can be *structural* or *statistical*. Information of the first type is typically used in the field of structural and syntactic pattern recognition (SSPR). Information of the second type is utilized in the field of statistical pattern recognition (SPR). In our study, we deal with the SPR approach.

The statistical information, or *features*, about the classes is typically represented by random vectors. The procedure for obtaining the features consists of mapping the feature values of each sample to a real-valued vector. Feature values, for example, can be the width or the height of a figure, the value of a pixel of an image, different measurements of graphical objects (such as the radius, texture, perimeter, area, smoothness, compactness), the pitch frequency of a speech utterance, etc.

SPR can also be subdivided into two well defined approaches, *parametric* SPR and *non-parametric* SPR. In the former, the random vectors have a fixed probability distribution, e.g. normal (or Gaussian), exponential, doubly exponential, multinomial, etc. The normal distribution assumption is the most widely used one in practical applications because of its general applicability and many important properties [47]. The statistical information about the classes is given by the parameters of the distribution: for example, the mean vector and the covariance matrix. These values can be learned by using any of the approaches known in the literature, such as *Bayesian learning* and the *maximum likelihood estimation* method [38, 47, 146].

In non-parametric SPR, methods such as the k-nearest neighbor rule or histogram methods are well-known, where the form of the distribution is not known or assumed. In our study, we solve open problems using parametric SPR techniques. One of these contributions is the formalization of the theory of optimal pairwise linear classifiers for normally distributed random vectors. These results are presented in Chapters 7 and 8 respectively. The other problem that we have solved using parametric SPR techniques is the development of the formal analysis of why heuristic functions work. This theory is presented in Chapter 9.

1.4 Research Problems in Pattern Recognition

1.4.1 Linear Classifiers

One of the problems we have studied involves the design of linear classifiers. Optimal classification is achieved by using a classifier called the Bayes classifier. Given c classes, $\omega_1, \ldots, \omega_c$, and an object to be classified, this object is assigned to the class that maximizes the *a posteriori* probability, which is calculated using Bayes theorem. When dealing with two Gaussian-distributed classes, the equality between their *a posteriori* probabilities leads to the discriminant function being a second-order polynomial. This polynomial can have the form of a quadratic function, or in particular, a linear function. In general, the discriminant function has the following form:

$$X^T A X + B^T X + c = 0\,, \tag{1.1}$$

where A is a real-valued matrix, B is a real-valued vector, c is a real number, and X is the feature vector representing the object to be classified.

When (1.1) has the form $B^T X + c = 0$, the function is linear. It represents a hyperplane in the d-dimensional space, and in particular, a straight line in the two dimensional space. For example, consider two normally distributed random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$, representing two classes, where M_i is the mean vector, and Σ_i is the covariance matrix, for $i = 1, 2$. Suppose that the parameters of $\mathbf{X}_1$ and $\mathbf{X}_2$ are the following:

$$M_1 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, M_2 \approx \begin{bmatrix} 4 \\ 2 \end{bmatrix}, \Sigma_1 = \Sigma_2 = \begin{bmatrix} 4.5 & 0.75 \\ 0.75 & 2 \end{bmatrix}.$$

The optimal classifier is linear, and given by the following linear equation:

$$-1.3037x + 2.48889y - 3.55556 = 0\,,$$

The two ellipses representing the classes, which we discuss later, and the corresponding optimal linear classifier are depicted in Figure 1.6.

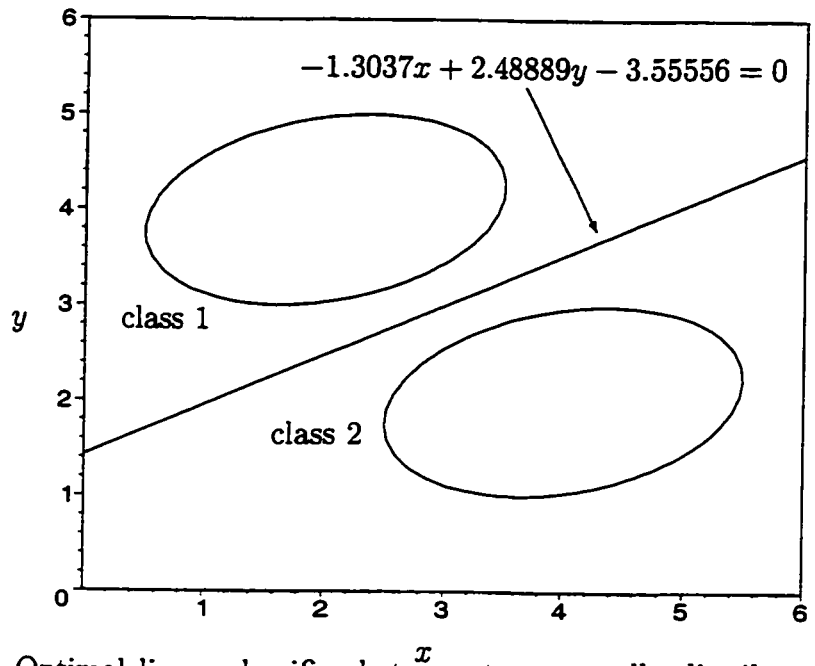


Figure 1.6: Optimal linear classifier between two normally distributed classes. The covariance matrices are equal.

Determining linear classifiers has been an extensively-studied problem for many years. Researchers have been studying them primarily because of their simplicity, and the corresponding speed of classification. This fact can be noted by comparing the form of the linear classifier (see also Figure 1.6), $B^T X + c = 0$, against that of the quadratic function given in (1.1). Many approaches to yield linear classifiers have been proposed in the literature, including *Fisher's approach* [38, 89, 146], the *perceptron algorithm* (the basis of the back propagation *neural network* learning algorithms) [87, 101, 122, 123], *piecewise recognition models* [119], *random search optimization* [121], *removal classification structures* [5], *adaptive linear dimensionality reduction* [88] (which outperforms Fisher's classifier for some data sets), and *linear constrained distance-based classifier analysis* [36] (an improvement to Fisher's approach designed for hyperspectral image classification). The major drawback of these approaches is the fact that the classifiers that they yield are not *optimal*.

1.4.2 Pairwise Linear Classifiers and Minsky's Paradox

As discussed earlier, when dealing with two normally distributed random vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, the optimal Bayes classifier is a quadratic function of the form in (1.1). More specifically, the quadratic term is of the form:

$$X^T(\Sigma_2^{-1} - \Sigma_1^{-1})X\,, \tag{1.2}$$

where Σ_1 and Σ_2 are the covariance matrices for $\mathbf{X}_1$ and $\mathbf{X}_2$, respectively.

It is well known that when $\Sigma_1 = \Sigma_2$, the term of (1.2) cancels, and hence the quadratic function of (1.1) becomes a linear function of the form $B^T X + c = 0$.

In this thesis, we demonstrate that there are other families of linear classifiers, namely those obtained by pairwise linear classification. Indeed, we have found that there are undiscovered cases in which the optimal classifier is linear, i.e. when the quadratic function given in (1.1) has the form of a pair of hyperplanes. We have derived the necessary and sufficient conditions to obtain the optimal pairwise linear classifier for two dimensional normally distributed random vectors. In this case, the discriminant function given in (1.1) can be written as follows:

$$(B_1^T X + c_1)(B_2^T X + c_2) = 0\,, \tag{1.3}$$

where B_1 and B_2 are real-valued vectors, and c_1 and c_2 are real numbers.

$B_1^T X + c_1$ and $B_2^T X + c_2$ are the linear factors of the second degree polynomial of (1.1). In particular, when these two factors are "coincident" (or identical), the optimal classifier is a single straight line, and this is the *only* scenario that has been reported in the literature. In our research, we have found the necessary and sufficient conditions for optimal pairwise linear classifiers for three cases.

In Section 1.4.1, we observed that the optimal Bayesian classifier can be either a quadratic or a linear function. In this section, we discuss a particular scenario of two classes, whose features are given by a 2-bit binary vector, such that the features of one

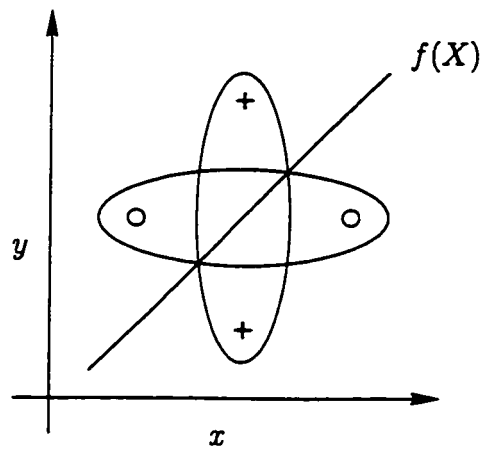


Figure 1.7: A simple scenario for the 2-bit binary-featured classes. Minsky showed that no single linear classifier can be found for these classes.

class are the exclusive OR of this vector, and the features of the other class are the exclusive NOR of the binary values. This particular scenario is illustrated in Figure 1.7. One of the classes is represented by the symbol '+', and the other class by the symbol 'o'.

In 1957, a famous mathematician, Marvin Minsky, showed that the *perceptron algorithm* could not find a single linear classifier for this simple scenario. Observe in Figure 1.7, that no linear function $f(X)$ correctly classifies objects of these classes. This classification problem is known as *Minsky's paradox* for the perceptron, or the *two-bit parity* problem [38]. The problem can easily be generalized for the k-bit parity problem.

The generalized scenario of this problem is that of two classes represented by binary d-dimensional random vectors for which no linear (hyperplane) classifier can be found. This problem, known as the *generalized Minsky's paradox*, has been unresolved for more than four decades. In Chapter 7, we present a solution to this problem for the two dimensional case by using *optimal pairwise linear* classifiers. In Chapter 8, we present the solution to this problem for the d-dimensional case. Not only do we resolve the paradox for the scenario when the features are binary vectors – we also resolve the paradox when the classes are normally distributed.

In particular, we have found that the optimal classifier is a pair of straight lines

when the covariance matrices have the form $\Sigma_1 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} b & 0 \\ 0 & a \end{bmatrix}$, and the mean vectors are equal. Hence, we have resolved the two dimensional generalized Minsky's paradox.

We have extended our formal analysis of the two-dimensional case, and determined the necessary and sufficient conditions for optimal pairwise linear classification between two d-dimensional normally distributed random vectors, where $d > 2$. Thus, we have resolved the *multi-dimensional* generalized Minsky's paradox also.

1.4.3 Heuristic Functions: Accuracy and Optimality

The second problem that we address using parametric SPR techniques is briefly mentioned in this section. One such problem concerns that of using *heuristics* to solve optimization problems. An optimization problem[2] is typically defined in terms of instances which are drawn from a (finite) set, $\mathcal{X}$, an objective function, and some feasibility functions. The aim is to find an (and hopefully, the unique) instance of $\mathcal{X}$, which leads to the maximum (or the minimum) value of the objective function subject to the feasibility constraints.

Consider the well-known Traveling Salesman Problem (TSP), in which the five cities are numbered from 1 to 5, and the salesman starts from city 1, visits every other city once, and returns to city 1. An instance of $\mathcal{X}$ is a permutation of the cities, for example, 1 4 3 2 5. The objective function for that instance, $f(1\,4\,3\,2\,5)$ is obtained by performing the summation of the inter-city distances: $1 \rightarrow 4$, $4 \rightarrow 3$, $3 \rightarrow 2$, $2 \rightarrow 5$, and $5 \rightarrow 1$. The optimal solution is the instance that minimizes the value of f.

A *heuristic algorithm* is an algorithm that attempts to find one or more instances of $\mathcal{X}$ that maximize f by iteratively invoking a *heuristic* function. The instance(s) that maximize(s) f will be the *solution(s)*[3] to the optimization problem. A *heuristic* is a method that performs one or more modifications to a given solution or instance, in order to obtain a different solution. The heuristic, in turn, invokes a heuristic function,

[2]Every optimization problem can also be formulated as a decision problem [49].

[3]We use the term "solution" to refer to an element $x \in \mathcal{X}$, and the term "profit" to refer to the value of $f(x)$. In minimization problems, $f(.)$ will be a cost funtion.

which estimates (or measures) the cost of the solution at the particular state in the search process.

Many heuristic algorithms and heuristic functions have been reported in the literature, where the former include the *alpha-beta search* [86], *backtracking*, *hill-climbing* [77], *simulated annealing* [1], *genetic algorithms* [95], *tabu search* [52], *learning automata* [102], etc. These heuristic algorithms can be used to solve a wide variety of NP-hard problems. Such problems can be found in a wide range of applications spanning the whole spectrum of artificial intelligence, and include game playing and game theory, graph theory, database query optimization, networking, computational geometry, number theoretic problems, parallel processing, etc.

One such problem is the classical n-puzzle problem [104]. This problem consists of a square board containing n square tiles and an empty position called the "blank". One of the better-known solutions to the n-puzzle problem is the A* algorithm and its variants (like the A÷ algorithm) [104, 140, 172]. This algorithm is a graph search algorithm that is used to find the path of minimum cost between two nodes, the start node and the goal node. The A* maintains a tree which stores the paths that are already explored. Using these paths, a measure, f, of the potential advantage of choosing each path is calculated. The value of f, which is the cost of traversing the graph between two nodes, can be computed (or rather, estimated) by using different heuristic functions. If the heuristic is "admissible", A* converges to the correct result. In this case, the heuristic function is an upper bound of the true cost from all nodes to the goal node.

In general, for any arbitrary problem, the question of how useful a heuristic function is, in determining the cost of traversing from one node to another, has no known analytic solution – it has traditionally been empirically analyzed. In this thesis, we present a formal analysis that provides a *stochastically positive* answer to the question of comparing the relative advantages of potential heuristic functions. This study will be the subject matter of Chapter 9.

1.5 Outline of the Thesis

This thesis is structured in four parts. The first part presents an overview of the thesis, and a description of the problems addressed in this research work. The second part includes an overview of data compression, and the problems that we have solved in this area, namely the enhanced version of static Fano coding, the greedy version of the adaptive Fano coding, and the adaptive version of Fano's method using binary search trees. The third part presents an overview of statistical pattern recognition and linear classifiers. It also includes the formal theory of optimal pairwise linear classifiers for normal distributions and the formal analysis of why heuristic functions work using doubly exponential and normal distributions. The last part of the thesis discusses the conclusions of this research work, and the future extensions. The content of each individual chapter is discussed below.

In Chapter 2, we present an overview of the state-of-art lossless data compression methods. In the first part of the chapter, the important concepts of information theory and coding theory are formalized. These include information content, entropy, properties of codes, and their mutual relationships. These concepts are also discussed for higher-order sources. The following sections include an overview of the static versions of coding methods such as Huffman's algorithm, Shannon's method, Fano's method, and arithmetic coding. They are discussed for the zeroth-order statistical model. Next, the adaptive version of Huffman's algorithm and arithmetic coding are presented. The issue that statistical coding methods can be improved by using higher-order source models is also discussed. In the interest of brevity, we present only the procedure for the Huffman coding. Finally, we introduce the state-of-art statistical models used by present-day encoding methods. They are the LZ family of algorithms, dynamic Markov compression (DMC), prediction with partial matching (PPM), the Burrows-Wheeler transform (BWT), and grammar-based coding (GBC), and other higher-order compression methods.

In Chapter 3, we present an enhanced version of the static Fano coding, which is based on the concept that we called "code word arrangement" of two lists. We discuss some properties of the Fano tree that are inherent to the foundations of a similar

structure, the Huffman tree. We then present the enhanced encoding and decoding algorithms, which are based on a rule that rearranges the code word in an increasing order of length, when the probabilities of the symbols are in the inverse order. We discuss some properties of the enhanced Fano encoding and decoding algorithms, its correctness and its prefix-code property. Finally, we present the empirical results that demonstrate the efficiency of the enhanced Fano coding on the files of two well-known benchmarks, the Calgary corpus and the Canterbury corpus. The preliminary version of these results has been published in the *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, held in Tucson, Arizona, USA, in October 2001 [133]. The corresponding journal paper is currently under review [132].

In Chapter 4, we present a greedy version of the adaptive Fano coding. We first present the adaptive encoding and decoding algorithms for the binary output alphabet, which utilize the nearly-equal-probability property. We then present a formal analysis for the correctness of the encoding and decoding algorithms, which achieve lossless and on-line compression. The computational worst-case complexity of the two algorithms is also discussed. The empirical results that we have conducted on the files of the Calgary corpus and the Canterbury corpus, and the comparison of the greedy approach introduced here with the traditional adaptive Huffman coding are also submitted. In the last part of this chapter, we discuss the greedy adaptive Fano coding for multi-symbol code alphabets, for which we present the corresponding nearly-equal-probability partitioning problem, and a greedy, sub-optimal partitioning procedure. A brief summary of this work will be presented at the 2002 IEEE Aerospace Conference, BigSky, MT, USA [130]. The corresponding journal paper is currently being reviewed [131].

Chapter 5 concerns an adaptive version of the Fano coding problem which uses a new family of trees called the Fano binary search tree (BST). We first present an overview of the BSTs and the heuristic used to adaptively maintain a BST. We then introduce two new families of BSTs, the partitioning BST and the Fano BST, the latter being a particular case of a partitioning BST with the added properties of Fano coding. We then present an algorithm that is used to generate a Fano BST for a particular source alphabet when the probabilities of the symbols are known. Using these concepts, we introduce the shift operators useful to update the tree so as to maintain a Fano BST,

namely, the Shift-To-Left and the Shift-To-Right operators. We formally prove the correctness of these two operators. We also discuss the conditional shifting heuristic, which is used to ensure that the updated tree satisfies the properties of Fano coding. The corresponding encoding and decoding algorithms that use the mentioned-above heuristic are also formalized in this chapter. We then present the formal analysis of the properties of the encoding and decoding algorithms, and their corresponding updating procedures. We finally give a formal analysis for the convergence of our new updating procedure. To demonstrate the power of our new scheme, we present the empirical results obtained on the files of the Calgary corpus and the Canterbury corpus. Numerical and graphical analyses for the adaptive version of the Fano coding using Fano BSTs are also discussed here. No results from this chapter have been published yet.

The rest of the thesis deals with our new results related to the field of pattern recognition.

In Chapter 6, we present an overview of statistical pattern recognition (SPR). Normally distributed random vectors and the concept of diagonalization are explained in the first part of the chapter. We then discuss the theory of pattern classification using formal Bayes decision theory, which maximizes the *a posteriori* probability. The required framework for the rest of this thesis are also introduced in this chapter.

In Chapter 7, we present the solution to one of the problems in the field of statistical pattern recognition, namely the problem of computing *pairwise* optimal linear Bayesian classifiers for two-dimensional normally distributed random vectors. The necessary and sufficient conditions for these classifiers are formalized for three cases. In particular, we propose the resolution of Minsky's paradox for the perceptron, which was unsolved for many decades. We also present some empirical results on synthetic data which demonstrate the very high accuracy obtained by using our classifiers. Finally, we present a formal model that uses the maximum likelihood estimate to approximate the parameters of normally distributed classes whose classifier is pairwise and linear. This solution leads to a constrained optimization problem, which is solved numerically. We have used the results to test the resulting classifier for real life medical diagnosis on the Wisconsin Diagnosis Breast Cancer data set. Our pairwise linear classifiers

yields a higher accuracy when compared with the traditional Fisher's linear classifier. The results from this chapter were published in the *Proceedings of the Joint IAPR International Workshops SSPR'2000 and SPR'2000* held in Alicante, Spain [138], and the full journal version will appear in the *IEEE Transactions on Pattern Analysis and Machine Intelligence* [136].

In Chapter 8, we extend the formal theory of optimal pairwise linear classifiers for multi-dimensional normally distributed random vectors. We generalize the analysis of Chapter 7 to yield the necessary and sufficient conditions required by the parameters of normal random vectors to yield the optimal classifier as a pair of hyperplanes. As before, three cases are analyzed in this chapter, the first being when the classes are diagonalized. The second is when the means are different and the covariance matrices satisfy certain conditions. The third case is when the mean vectors are coincident and the covariance matrices satisfy certain conditions, which represents the generalized Minsky's paradox for the perceptron. The chapter also discusses the empirical results that we have conducted on synthetic data for the three different cases analyzed in the chapter. A brief survey of the results presented in this chapter was published in the *Proceedings of the 2001 Joint Conference on Artificial Intelligence* held in Adelaide, Australia in December 2001 [139]. The corresponding journal paper is currently under review [139].

In Chapter 9, we formally analyze the problem of why heuristic functions work. We first give an overview of various related open, optimization problems in computer science, and the application of heuristic algorithms and heuristic functions used to solve these open problems. We present a formal analysis that relates the efficiency and optimality of heuristic algorithms that utilize different heuristic functions. We then submit an analysis for this scenario using doubly-exponentially distributed random variables and pattern classification techniques. We continue to give the corresponding numerical analysis for normally distributed random variables. Finally, we confirm these results by reporting the empirical results that we have obtained from our simulations on database query optimization problems. Part of the work from this chapter was published in the *Proceedings of the 7th International Conference on Database Systems for Advanced Applications*, held in Hong Kong in April 2001 [105]. The corresponding

journal paper is currently under review [112].

Chapter 10 summarizes the contributions of our research work. Future potentials and extensions of the models that we have studied are also mentioned in this chapter.

Part II

Data Compression: Overview and Research

Chapter 2

Data Compression

Data compression is not a new idea. It dates back form the history of Roman civilization, where a shorthand form of writing was designed. In the nineteenth century, Morse developed an efficient code used to transmit messages, which nowadays is called the Morse code. In the same century, Braille created a writing system for blind people. This system consists of encoding a symbol of the alphabet as a set of points which can be recognized by a blind person by using his/her fingers. The most amazing characteristic of this system was the fact that shorter codes were assigned to more frequent symbols, e.g. the symbol 'e'. Most of the current statistical methods use this property.

In general, data compression can be thought of a process of transforming an input stream of data, producing (if efficient) a shorter one. The inverse process is applied to the compressed data obtaining a new stream of data. If the latter is *exactly* the same as the initial stream of data, the process is called *lossless data compression*. On the contrary, if the new stream is different than the original input, we are talking about *lossy data compression*. Both lossless and lossy data compression are widely used. Lossy compression techniques and applications can be found in [25, 33, 114, 120, 145].

Data compression has applications in many fields, such as communications and storage. Most of data compression methods are based on formal concepts of *coding theory* and *information theory*. Although information theory was first developed for communication systems, it has been widely and successfully applied to the storage applications of data compression too. In the following sections, we discuss how both

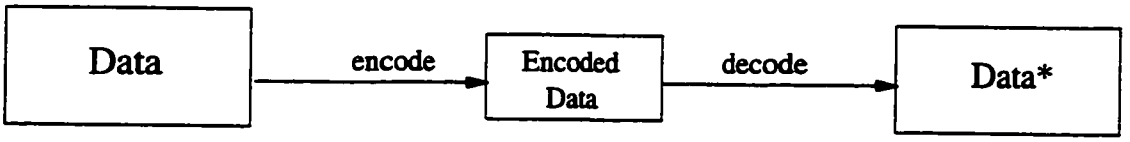


Figure 2.1: Data Encoding and Decoding.

theories are strongly related, and how they constitute a firm basis for data compression.

2.1 Coding Theory

2.1.1 An Encoding System

As discussed above, data compression consists of the process of transforming an input stream of data, producing a smaller one. Based on this idea, an encoding system can be defined as a similar process, in which we have the input stream of data as well, and we want to produce another stream of data. Such a process is depicted in Figure 2.1. A data compression system can be seen as a particular case of the encoding system. *Data** is recovered from the encoded data. If *Data* and *Data** are equal, the data compression process is lossless.

Another particular case of an encoding system is an error detection-correction system. The main purpose of this system is to ensure the correctness of the data recovered after decoding. Unlike the data compression system, this system introduces extra information in the encoded data, generating a larger output. These techniques are discussed in detail elsewhere [56, 57, 58, 62, 117, 127].

All the systems discussed above are based on the idea of a communication system, discussed in [57], in which the encoded data is a message to be send through a channel. The decoder receives the message from the channel and produces an output stream of data.

2.1.2 Sources

Some of the concepts mentioned in the previous subsections are more formally defined here, some of them are concepts whose definitions can be found in [58, 127]. We have

talked about an input stream of data, which is composed of symbols generated by a source defined as follows.

Definition 2.1. A *source* is an ordered pair $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ (the alphabet) and $\mathcal{P}$ (a probability distribution on $\mathcal{S}$) are defined below.

Definition 2.2. The *source alphabet*, $\mathcal{S}$, is defined as the set of symbols $\{s_1, \ldots, s_m\}$, which the input sequence can contain. The cardinality of $\mathcal{S}$ is m, also referred to as $|\mathcal{S}|$.

Definition 2.3. Given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the probability of occurrence the symbol s_i, p_i (or $\Pr[s_i]$), is defined as the probability of randomly selecting s_i from $\mathcal{S}$. The probabilities for all the symbols of $\mathcal{S}$ are referred to as the vector $\mathcal{P} = [p_1, \ldots, p_m]$.

The probability of occurrence of a symbol will be synonymously referred to as frequency, relative frequency, or probability of occurrence.

In this report, we use the symbols $\mathcal{S}$ (which corresponds to the source alphabet) to refer to either the source alphabet or the source itself.

The input sequence of the encoding process is defined as follows.

Definition 2.4. Given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, a *source sequence* (or *input sequence*), $\mathcal{X}$, is the string $x[1] \ldots x[M]$, such that for $i = 1 \ldots M$, $x[i] \in \mathcal{S}$.

A source is said to be *stationary* if any sequence that it generates satisfies:

$$\Pr[x[i] \ldots x[i+k]] = \Pr[x[i+t] \ldots x[i+k+t]], \tag{2.1}$$

for any integer t.

This property is given in terms of stationarity in a *weak sense* and is sufficient for our work. The actual definition of stationarity in the *wide sense* is quite stronger and can be found in [70, 113].

A source is said to be *memoryless* or *zeroth-order* if for all $i, j = 1, \ldots, m$, $i \neq j$, p_i is independent of p_j.

Given these definitions above, we use the notation that $\mathcal{S}^+$ is the set of all possible source sequences, and every $\mathcal{X} \in \mathcal{S}^+$.

We have a source sequence, $\mathcal{X}$, which we want to transform into another sequence, $\mathcal{Y}$, by the encoding process. The output of the encoder must be something that the receiver is able to understand. In real situations, the receiver is a communication channel that accepts only binary digits or an operating system that interprets only ASCII characters. The set of symbols that the receiver understands is called the code alphabet, and is defined as follows:

Definition 2.5. The *code alphabet*, $\mathcal{A}$, is defined as the set of symbols $\{a_1, \ldots, a_r\}$ contained in the output sequence. The cardinality of $\mathcal{A}$ is n, also referred to as $|A|$.

The output of the encoding process in Figure 2.1 is defined as the *output sequence*, which is composed of symbols drawn with repetition from the code alphabet, $\mathcal{A}$.

Definition 2.6. Given the code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$, an output sequence, $\mathcal{Y}$, is the string $y[1] \ldots y[R]$, such that for $j = 1 \ldots R$, $y[j] \in \mathcal{A}$.

Definition 2.7. Given the code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$, a code word, w_i, is given by the sequence $w_{i_1} \ldots w_{i_{\ell_i}}$, such that for $j = 1 \ldots \ell_i$, $w_{i_j} \in \mathcal{A}$, or simply $w_i \in \mathcal{A}$. The length of w_i is given by ℓ_i.

From Definitions 2.5 and 2.7, we observe that $\mathcal{A}^+$ is the set of all possible output sequences and every $\mathcal{Y} \in \mathcal{A}^+$. The process of transforming an input sequence into an output sequence utilizes an *encoding function* defined as follows:

Definition 2.8. Given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, and the code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$, an *encoding function*, ϕ, is defined as the function $\phi : \mathcal{S}^+ \rightarrow \mathcal{A}^+$.

The function ϕ determines a *code*. Some properties that this function and the code must satisfy are discussed in the next section. The association of a code word with a symbol of $\mathcal{S}$ is called *encoding scheme*.

Definition 2.9. Given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, an *encoding scheme* is defined as the list of productions:

$$\begin{aligned} \phi: \quad & s_1 \rightarrow w_1 \\ & \vdots \\ & s_m \rightarrow w_m, \end{aligned}$$

where $w_i \in \mathcal{A}^+$, for $i = 1, \ldots, m$.

The encoding function ϕ consists of the concatenation of code words obtained by replacing each symbol of the input sequence, $\mathcal{X}$, by its correspondent code word, according to the encoding scheme given in Definition 2.9. The concatenation of code words is done with no separating spaces, commas or other marks. Some of the encoding methods discussed later in this chapter are based on this concept. These methods are also called *encoding by replacement scheme* methods.

Example 2.1. Suppose that the source alphabet is $\mathcal{S} = \{a, b, c, d\}$ and the code alphabet is $\mathcal{A} = \{0, 1\}$. An example of an encoding scheme is given the following list of productions:

$\phi : a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, and $d \rightarrow 111$.

Suppose also that we have the following input sequence $\mathcal{X} = ccad$. The encoding function of $\mathcal{X}$ is obtained as a concatenation of code words according to the encoding scheme ϕ. That is, the output generated by the encoding function of $\mathcal{X}$ is $\phi(\mathcal{X}) = \phi(ccad) = 1101100111 = \mathcal{Y}$.

□

2.1.3 Codes

In the previous subsection, we defined an encoding function and an encoding scheme. We now use another related concept, the *code*, which we define as $\mathcal{C} = \{w_1, \ldots, w_m\}$, where w_i is a code word, for $i = 1, \ldots, m$.

In Example 2.1, we observe that the code words in the list of productions do not have the same length. In this case, the code is a *variable-length code*. Conversely, a code in which all the code words have the same length is classified as a *fixed-length code*.

Definition 2.10. Consider the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. The code word w_i, whose length is ℓ_i, is *prefix* of the code word w_j, whose length is ℓ_j, if $\ell_i \leq \ell_j$ and, for $k = 1 \ldots \ell_i$, $w_{i_k} = w_{j_k}$.

Example 2.2. Suppose that the source alphabet is $S = \{a, b, c\}$, and the code alphabet is $X = \{0, 1\}$. Consider also the encoding scheme: $\phi : a \to 0$, $b \to 00$, and $c \to 10$.

The code word 0 is prefix of 00, since they agree in the first position and the length of 0 is 1. □

One of the most important properties of a code is the *prefix condition*, which states that none of the code words is prefix of any other code word. More formally,

Definition 2.11. Consider the encoding scheme $\phi : s_i \to w_i$, for $i = 1, \ldots, m$. The code determined by ϕ, $\mathcal{C}$, satisfies the *prefix condition* if, for $i, j = 1, \ldots, m$, $i \neq j$, w_i is not prefix of w_j.

For example, the code determined in Example 2.1 is a prefix code. The code of Example 2.2 is not a prefix code, since the code word 0 is prefix of the code word 00.

A code that satisfies the prefix condition is referred to as a *prefix code*[1]. Although this name is sometimes misleading[2], we refer to a *prefix code* a code that satisfies the prefix condition.

In lossless data compression situations, we want to encode a stream of data, and to later recover the same stream of data "exactly". While the prefix condition imposes some restrictions in the encoding scheme, we want some restrictions in the whole stream of data. That is, we are looking for some relation between the input sequence and the output sequence obtained by the encoding function. The restriction is the *uniquely decodability* property of a code, which states that for *every* output sequence in $\mathcal{A}^+$ obtained by the encoding function there exists *one and only one* source sequence in $\mathcal{S}^+$.

[1]A prefix code is also called an instantaneous code. The reason of this name is the fact that the decoder can "instantaneously" recognize source symbols as it receives each code symbol from the output sequence.

[2]The fact that a prefix code is a code in which "no" codeword is prefix of any other code word is confusing, but is widely used in the literature.

Definition 2.12. Consider the encoding function $\phi : \mathcal{S}^+ \to \mathcal{A}^+$. The code determined by ϕ is *uniquely decodable* if and only if ϕ is an injection.

Till now, we have spoken about some properties of the codes required for lossless compression. We shall now speak about how to build a code that satisfies these conditions.

Theorem 2.1 (Kraft's Inequality). Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and a set of positive integers $\{\ell_1, \ldots, \ell_m\}$. There exists a *prefix* encoding scheme given by the list of productions $s_i \to w_i$, if and only if

$$\sum_{i=1}^{m} r^{-\ell_i} \leq 1 . \tag{2.2}$$

□

The proof of Theorem 2.1, can be found in [3, 32, 58]. Kraft's inequality gives some restrictions about the lengths of the code words, but it does not say how to build the code once we have the length for each code word. We now proceed from just considering the length of each individual code word in the encoding scheme. In fact, what we need is that ϕ be an injection. The following theorem states some conditions in order to satisfy this restriction.

Theorem 2.2 (McMillan's Inequality). Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet and $\mathcal{A} = \{a_1, \ldots, a_r\}$ be the code alphabet. If the code determined by the list of productions $s_i \to w_i$ is uniquely decodable, then

$$\sum_{i=1}^{m} r^{-\ell_i} \leq 1 .$$

□

The main difference between Kraft's inequality and McMillan's inequality is that the former imposes a condition on a code to be *prefix* (on individual code words),

while in the latter the condition is imposed on an encoding function, $\phi : \mathcal{S}^+ \rightarrow \mathcal{A}^+$, (i.e. on the entire output sequence). In fact, a prefix code ensures that the encoding function generates a uniquely decodable code, and hence the former is the necessary and sufficient condition for the unique decodability property.

The proof of Theorem 2.2 can be found in [58]. If we have a uniquely decodable code, we ensure lossless compression. In fact, with a prefix code we ensure unique decodability. As a consequence all we need is to find a prefix code. However, as explained till now, we only know restrictions about the lengths of the code words, but have developed no procedure to build a prefix code. This "hard" problem is well solved by well-known encoding methods that will be discussed later in this chapter.

2.2 Information Theory

2.2.1 Introduction

The formal basis of information theory was introduced by C. Shannon in 1948 when he published his paper titled "A Mathematical Theory of Communication" [147]. Although this theory includes concepts applied to communication systems, these concepts can be applied to data storage management, and in general, to data compression.

This theory is based on formal mathematics concepts used to measure the amount of information to be transmitted or stored. The use of logarithmic functions in the definition of the amount of information has some important advantages. The logarithm of a product is easily converted into a sum of logarithms and the logarithm of a division can be converted into a difference of logarithms. These relations are discussed in the next section.

2.2.2 Information

We have talked about input sequences, which are composed of symbols drawn from a finite source alphabet, $\mathcal{S}$. In the real life, an input sequence represents a message to be transmitted between two points, a file to be stored in a hard disk, etc. This data is generated by human beings in different languages or environments. Some environments

make use of some symbols more frequently than others. For example, in English, the letter e is the most likely one to appear in a text, and the letter z is one of the least likely characters. If we are managing the data processed by a bank, it is clear that numbers are more likely, and the number 0 may be the most likely one.

We now introduce an important concept in information theory, the information amount of a symbol.

Definition 2.13. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$. The *information amount* contained in s_i is given by:

$$\mathcal{I}(s_i) = \log_r \frac{1}{p_i} = -\log_r p_i \,, \tag{2.3}$$

where p_i is the probability of occurrence of the symbol s_i.

This powerful definition has different interpretations in real life. One of them is related to the fact that the higher the probability of a symbol, the smaller the information amount. By convention, when $p_i = 1$, $\mathcal{I}(s_i) = 0$. For example, suppose that we read a text in English and we found a z. If we consider the event "a z occurs in the text" with a small probability, the information amount of this event is high. Therefore, if we see many z's in the text, we will be very surprised.

Another interpretation is related to the base of the logarithm that we choose. $\mathcal{I}(s_i)$ is measured in *r-ary* units. When $r = 10$, the information amount is measured in Hartleys, and $-\log_{10} p_i$ is the number of decimal digits needed to encode s_i. When $r = 2$, the information amount is measured in bits, and $-\log_2 p_i$ is the number of binary digits (bits) needed to encode s_i.

The second interpretation is very important, since it establishes the relationship between information theory and coding theory. From now on, when we use the information amount of a symbol we will not specify the base of the logarithm. We assume that the base is r, surprisingly, the cardinality of the code alphabet set. This is not a casual coincidence, because if, for example, our code alphabet is $\mathcal{A} = \{0, 1\}$, the information amount is measured in bits and we have to represent our source alphabet symbols using these binary digits.

Example 2.3. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$.

The amounts of information contained in the source alphabet symbols are: $\mathcal{I}(a) = -\log .4 \approx 1.32$, $\mathcal{I}(b) = -\log .3 \approx 1.74$, $\mathcal{I}(c) = -\log .2 = 2.32$, and $\mathcal{I}(d) = -\log .1 \approx 3.32$.

□

2.2.3 Entropy

Entropy is the most important concept in information theory. It represents the average information content of a source, given the probabilities of the symbols.

Definition 2.14. Consider a stationary, memoryless source whose alphabet is $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities $\mathcal{P} = [p_1, \ldots, p_m]$. The average information amount (or entropy) of $\mathcal{S}$ is given by:

$$\mathcal{H}(\mathcal{S}) = \sum_{i=1}^{m} p_i \mathcal{I}(s_i) = -\sum_{i=1}^{m} p_i \log p_i \,. \tag{2.4}$$

As a definition of information content, entropy has different interpretations. In the case of the binary alphabet, entropy can be interpreted as the average number of bits needed to encode an input sequence of symbols drawn from a source alphabet with the underlying probabilities of occurrence. This concept is very important in coding theory, since it establishes a bound in coding.

One of the most important properties of entropy is that it is positive and no greater than the logarithm of the cardinality of the code alphabet set. In other words, given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$ and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, the entropy of $\mathcal{S}$, obtained as in Definition 2.14, satisfies:

$$0 \leq \mathcal{H} \leq \log m. \tag{2.5}$$

The proof of Equation (2.5) can be found in [53, 58].

In Section 2.1, we discussed various aspects of code words and their lengths. We also know that the source alphabet symbols occur with specific probabilities, and hence we can calculate the average length of an encoding scheme (or code) as follows:

Definition 2.15. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and the encoding scheme $\phi : s_i \rightarrow w_i$. The average code word length of ϕ is given by:

$$\bar{\ell} = \sum_{i=1}^{m} p_i \ell_i \,, \tag{2.6}$$

where ℓ_i is the length of w_i.

Observe that by definition, the main goal of data compression is to encode data into a more compact form. The smaller the average length of the code, the shorter the encoded data, and hence the better the compression. In other words, we would like to minimize $\bar{\ell}$, and we simultaneously need a prefix code. This hard problem is well solved by encoding methods discussed later, but the question is one of determining how much we can reduce $\bar{\ell}$. The answer to this question is given in the following theorem, stated and proved by C. Shannon [147].

Theorem 2.3 (Noiseless coding theorem). Consider a stationary, memoryless source with alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, whose probabilities are $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Suppose that the entropy of the source alphabet, $\mathcal{H}$, is calculated by using (2.4). For every prefix encoding scheme $\phi : w_i \rightarrow s_i$, we have:

$$\mathcal{H} \leq \bar{\ell}, \tag{2.7}$$

where $\bar{\ell}$ is calculated by using (2.6), and there exists at least one prefix encoding scheme that satisfies

$$\bar{\ell} < \mathcal{H} + 1 . \tag{2.8}$$

□

The proof of Theorem 2.3 can be found in [3, 58]. This theorem is one of the fundamental theorems in information theory. Equation (2.7) states that the lower bound for the average code word length of any prefix code is $\mathcal{H}$.

Example 2.4. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$. Suppose the following prefix encoding scheme: $\phi : a \rightarrow 0$, $b \rightarrow 10$, $c \rightarrow 110$, and $d \rightarrow 111$.

The average code word length is $\bar{\ell} = (.4)1 + (.3)2 + (.2)3 + (.1)3 = 1.9$.

The entropy of $\mathcal{S}$ is $\mathcal{H} = -.4 \log .4 - .3 \log .3 - .2 \log .2 - .1 \log .1 \approx 1.85$.

Observe that the code is prefix and that the inequalities given in Theorem 2.3 are satisfied. That is,

$$1.85 \approx \mathcal{H} \leq \bar{\ell} = 1.9 < \mathcal{H} + 1 \approx 2.85 .$$

□

2.2.4 Higher-order Sources

In Section 2.1.2, we defined a memoryless source, in which the occurrence of each symbol is statistically independent of the occurrence of any other symbol. When the probability of each symbol depends on the k previous symbols, the source is called a kth*-order* or *higher-order* source. When we consider a model in which the source is non-zeroth-order, this model will be referred to as a higher-order model.

On the other hand, the source alphabet, composed of all possible words of length k, $s_{i_1} \ldots s_{i_k}$ and obtained by concatenating the symbols of $\mathcal{S}$, is known as the kth *extension* of the zeroth-order source, $\mathcal{S}^k$ [3, 56, 57]. The probability of occurrence of $s_{i_1} \ldots s_{i_k}$ is given by $p_{i_1} \ldots p_{i_k}$.

The kth-order source is referred to as $\mathcal{S}^{(k)}$. The probability of occurrence of a symbol s_j, given $s_{i_1} \ldots s_{i_k}$, is calculated, by the law of conditional probabilities, as follows:

$$p(s_j|s_{i_1}\dots s_k) = \frac{p_{i_1,\dots i_k,j}}{p_{i_1,\dots i_k}}, \tag{2.9}$$

where $p_{i_1}\dots p_{i_k}$ is the probability of occurrence of $s_{i_1}\dots s_{i_k}$.

Equation (2.9) is the probability that the symbol s_j occurs given the occurrence of $s_{i_1}\dots s_{i_k}$. The sequence $s_{i_1}\dots s_{i_k}$ is also referred to as a *context*. The kth-order source, $\mathcal{S}^{(k)}$, has m^k different contexts.

The sequence $s_{i_1}\dots s_{i_k}s_j$ can be considered as a random vector whose relative frequency is $p_{i_1,\dots i_k,j}$. Hence, the probability of s_j can be calculated as the marginal probability of a random vector:

$$p_j = \sum_{i_1,\dots,i_k=1}^{m} p_{i_1,\dots i_k,j}. \tag{2.10}$$

By the law of total probabilities, the following equation is satisfied:

$$\sum_{j=1}^{m} p_j = \sum_{i_1,\dots,i_k,j=1}^{m} p_{i_1,\dots i_k,j} = 1 \tag{2.11}$$

As in the zeroth-order source, the entropy of the kth-order source is defined as follows.

Definition 2.16. Consider the kth-order source, $\mathcal{S}^{(k)}$. The kth-order average information amount (or entropy) of $\mathcal{S}^{(k)}$ is given by:

$$\mathcal{H}^{(k)}(\mathcal{S}) = -\sum_{i_1,\dots,i_k=1}^{m} p_{i_1,\dots,i_k} \sum_{j=1}^{m} p(s_j|s_{i_1}\dots s_{i_k}) \log p(s_j|s_{i_1}\dots s_{i_k}) \tag{2.12}$$

From the previous definition, we conclude that the entropy of $\mathcal{S}^{(k)}$ is the average of the average information content over the m^k different contexts. The average code

word length of the kth-order source can also be defined as follows.

Definition 2.17. Consider the kth-order source $\mathcal{S}^{(k)}$ whose probabilities of occurrence are given by $p(s_j|s_{i_1}\dots s_{i_k})$, $i_1,\dots,i_k,j = 1,\dots,m$, and the code alphabet $\mathcal{A} = \{a_1,\dots,a_r\}$. Suppose that $\ell(s_j|s_{i_1}\dots s_{i_k})$ is the length of $w(s_j|s_{i_1}\dots s_{i_k})$ obtained by means of the encoding scheme $\phi : s_j|s_{i_1}\dots s_{i_k} \to w(s_j|s_{i_1}\dots s_{i_k})$. The average code word length of ϕ is given by:

$$\bar{\ell}^{(k)} = \sum_{i_1,\dots,i_k=1}^{m} p_{i_1,\dots,i_k} \sum_{j=1}^{m} p(s_j|s_{i_1}\dots s_{i_k})\ell(s_j|s_{i_1}\dots s_{i_k}), \tag{2.13}$$

where $\ell(s_j|s_{i_1}\dots s_{i_k})$ is the length of the code word assigned to s_j given the context $s_{i_1}\dots s_{i_k}$.

As in the entropy definition, the average code word length of a code for $\mathcal{S}^{(k)}$ is the average of the average code word lengths for the m^k different contexts.

The noiseless coding theorem is also valid for the kth-order source.

Theorem 2.4 (Noiseless k^{th}-order coding theorem). Consider a kth-order source, $\mathcal{S}^{(k)}$, whose probabilities are given by $p(s_j|s_{i_1}\dots s_{i_k})$, $i_1,\dots,i_k,j = 1,\dots,m$, and the code alphabet $\mathcal{A} = \{a_1,\dots,a_r\}$. Suppose that the entropy of the kth-order source, $\mathcal{H}^{(k)}$, is calculated by using (2.12). For every prefix encoding scheme $\phi : s_j|s_{i_1}\dots s_{i_k} \to w(s_j|s_{i_1}\dots s_{i_k})$:

$$\mathcal{H}^{(k)} \le \bar{\ell}^{(k)}, \tag{2.14}$$

where $\bar{\ell}^{(k)}$ is calculated by using (2.13), $\ell(s_j|s_{i_1}\dots s_{i_k})$ is the length of the code word $w(s_j|s_{i_1}\dots s_{i_k})$, and there exists at least one prefix encoding scheme that satisfies

$$\bar{\ell}^{(k)} < \mathcal{H}^{(k)} + 1. \tag{2.15}$$

□

The proof of Theorem 2.4 can be found in [58].

Example 2.5. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ and the code alphabet $\mathcal{A} = \{0, 1\}$. Suppose that the probabilities of occurrence of $\mathcal{S}^{(1)}$ are given by the following matrix:

$$[p_{ij}] = \begin{bmatrix} .2 & .1 & .05 & .05 \\ .06 & .15 & .06 & .03 \\ .1 & .03 & .06 & .01 \\ .04 & .02 & .03 & .01 \end{bmatrix},$$

where the element in the ith row and j^{th} column is the probability of $s_i s_j$; e.g. $p_{ab} = .06$.

The probabilities of $\mathcal{S}$, obtained by using (2.10), are $[.4, .3, .2, .1]$.

The conditional probabilities can be calculated by using (2.9); e.g. $p(a|b) = \frac{p_{ba}}{p_b} = \frac{.06}{.3} = .2$. The conditional probabilities of $\mathcal{S}^{(k)}$ are given in the following matrix:

$$\mathcal{P}^{(1)} = \begin{bmatrix} .5 & .25 & .125 & .125 \\ .2 & .5 & .2 & .1 \\ .5 & .15 & .3 & .05 \\ .4 & .2 & .3 & .1 \end{bmatrix}.$$

Let us propose encoding schemes for $\mathcal{S}^{(1)}$:

$s_j \to w(s_j|a) : \{0, 10, 110, 111\}$, $s_j \to w(s_j|b) : \{110, 0, 10, 111\}$,

$s_j \to w(s_j|c) : \{0, 110, 10, 111\}$, and $s_j \to w(s_j|d) : \{00, 01, 10, 11\}$.

In this example, there are four different contexts, and hence four encoding schemes. In general, for a kth-order source, there are m^k different encoding schemes. The procedure for obtaining these encoding schemes is discussed in Section 2.6.

The entropy of the kth-order source, $\mathcal{S}^{(1)}$, is:

$$\begin{aligned}
\mathcal{H}^{(1)} &= 4[(.5)\log .5 + (.25)\log .25 + (.125)\log .125 + (.125)\log .125] + \\
&\quad .3[(.2)\log .2 + (.5)\log .5 + (.2)\log .2 + (.1)\log .1] + \\
&\quad .2[(.5)\log .5 + (.15)\log .15 + (.3)\log .3 + (.05)\log .05] + \\
&\quad .1[(.4)\log .4 + (.2)\log .2 + (.3)\log .3 + (.1)\log .1] \\
&\approx 1.74
\end{aligned}$$

The average code word length of the kth-order source, $\mathcal{S}^{(1)}$, encoded by the encoding schemes proposed above, is:

$$\begin{aligned}
\bar{\ell}^{(1)} &= .4[(.5)1 + (.25)2 + (.125)3 + (.125)3] + .3[(.2)3 + (.5)1 + (.2)2 + (.1)3] + \\
&\quad .2[(.5)1 + (.15)3 + (.3)2 + (.05)3] + .1[(.4)2 + (.2)2 + (.3)2 + (.1)2] \\
&= 1.78
\end{aligned}$$

As stated in Theorem 2.4, $1.74 \approx \mathcal{H}^{(1)} \leq \bar{\ell} = 1.78 < \mathcal{H}^{(1)} + 1 \approx 2.74$. □

2.3 Compression Measures

In the current bibliography, we can find different ways of measuring the compression efficiency. For the sake of clarity, we believe that it is useful to dedicate a short section to defining these different measures. These measures define a relation between the size of the original data (the input sequence) and the size of the encoded data (the output sequence). In most cases, the source alphabet and the code alphabet are different, and hence the cardinality of their respective sets could also be different. The main question that arises at this point is the following: "how do we compare 'apples' and 'oranges'?".

The answer to this question is to define an average code word length based on the same alphabet. We have already defined the average code word length of an encoding scheme. The input sequence has to be defined by means of code alphabet symbols and then *parsed* into code words (source alphabet symbols) satisfying the *strong parsing property* [58], defined as follows.

Definition 2.18. Consider the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. The set of r-ary input sequences, $\{s_1, \ldots, s_m\}$ has the *strong parsing property* if and only if every r-ary input sequence word $\mathcal{X} \in \mathcal{A}^+$ is *uniquely* a concatenation: $\mathcal{X} = s_{i_1} \ldots s_{i_t} v$, such that $\{s_1, \ldots, s_m\}$ satisfies the prefix condition and $\text{length}(\mathcal{X}) < \max_{1 \le i \le m}\{\text{length}(s_i)\}$.

From this definition, the length of the source symbol s_i, l_i, is the number of the code alphabet symbols that s_i contains. We can now define the average source symbol length of $\mathcal{S}$ as follows.

Definition 2.19. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, such that s_i has l_i code alphabet symbols. The *average source symbol length*, $\bar{l}$, is

$$\bar{l} = \sum_{i=1}^{m} p_i l_i \quad r\text{-ary digits.} \tag{2.16}$$

Example 2.6. Suppose we have the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the binary code alphabet $\mathcal{A} = \{0, 1\}$. The source symbols $a = 00$, $b = 01$, $c = 10$, and $d = 11$ satisfy the strong parsing property.

Let us calculate the average source symbol length:

$\bar{l} = 2(.4) + 2(.3) + 2(.2) + 2(.1) = 2(.4 + .3 + .2 + .1) = 2$ bits.

We observe that $\bar{l} = 2 = \log_2 4$. This is not a coincidence. In general, for a fixed-length source alphabet $\bar{l} = \log_r m$, where $|\mathcal{S}| = m$ and $|\mathcal{A}| = r$. □

In general, the information is represented by using the ASCII code which is internally stored as binary symbols (bits). The source alphabet size is $m = 256$, the code alphabet size is $r = 2$, and the average source symbol length is $\bar{l} = \log_2 256 = 8$ bits.

With these definitions clarified, we can discuss the different compression measures. We define three of them in this section: the *compression ratio*, the *percentage of compression*, and the *percentage of the remaining*. They represent relations between the sizes of the original data (the input sequence) and the encoded data (the output sequence).

Suppose that we have an input sequence, $\mathcal{X}$, whose length is $l_{\mathcal{X}}$, and, after encoding, we obtain an output sequence, $\mathcal{Y}$, whose length is $\ell_{\mathcal{Y}}$. We need to measure the length of $\mathcal{X}$ in code alphabet symbols. Observe that by the law of large numbers, the expected source sequence length (in code alphabet symbols) of $\mathcal{X}$ is $l_{\mathcal{X}}\bar{l}$. The output sequence, $\mathcal{Y}$, is obtained by replacing each symbol of $\mathcal{X}$ by its corresponding code word in the encoding scheme. Since the expected code word length is $\bar{\ell}$, the expected output sequence length is $l_{\mathcal{X}}\bar{\ell}$.

Definition 2.20. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Suppose that the output sequence, $\mathcal{Y}$, whose length is $\ell_{\mathcal{Y}}$, is obtained by encoding the input sequence, $\mathcal{X}$, whose length is $l_{\mathcal{X}}$, based on the encoding scheme $\phi : s_i \to w_i$. Suppose also that the average code word of this encoding scheme is $\bar{\ell}$ r-ary digits and the average input sequence length is $\bar{l}$ r-ary digits.

The *compression ratio*, φ, is given by the equation

$$\varphi = \frac{l_{\mathcal{X}}}{\ell_{\mathcal{Y}}} \approx \frac{\bar{l}}{\bar{\ell}}. \tag{2.17}$$

The *percentage of compression*, ρ, is given by the equation:

$$\rho = (1 - \frac{\ell_{\mathcal{Y}}}{l_{\mathcal{X}}})100 \approx (1 - \frac{\bar{\ell}}{\bar{l}})100\,. \tag{2.18}$$

The *percentage of the remaining*, δ, is given by the equation

$$\delta = \frac{\ell_{\mathcal{Y}}}{l_{\mathcal{X}}}100 \approx \frac{\bar{\ell}}{\bar{l}}100\,. \tag{2.19}$$

The compression ratio is a positive real number, φ, such that $0 < \varphi < \infty$. In order to achieve a value of φ equal to 0, $\bar{\ell}$ must be ∞, and for a value of ∞, $\bar{\ell}$ must be 0. Both cases are theoretically impossible. Infinite compression causes the original data to disappear and infinite expansion causes an endless output sequence. When $\varphi = 1$,

no compression is achieved. In all cases, compression is achieved whenever $\varphi > 1$, and the higher the compression the larger the value of φ.

The percentage of compression represents the amount, in percentage, that the output sequence is reduced with respect to the size of the input sequence.

The percentage of compression is a real value, ρ. When $\rho = 0$, no compression is achieved. When $0 < \rho < 100$, it means that some compression is achieved, and as $\rho \to 100$ the compression is better. Again, there is a theoretical obstacle for $\rho = 100$, which means infinite compression. When $\rho < 0$, the encoding process causes expansion.

The percentage of the remaining represents the proportion, in percentage, of the output sequence with respect to the input sequence.

The percentage of the remaining is a real value, δ. When $\delta = 0$, the compression is infinite, which is theoretically impossible. When $\delta = 100$, no compression is achieved. The smaller the value of δ, the higher the compression. When $\delta > 100$, the encoding process causes expansion.

Example 2.7. Consider the source alphabet, the probabilities of occurrence, and the code alphabet given in Example 2.6. Suppose that we have the following prefix encoding scheme: $\phi : a \to 0$, $b \to 10$, $c \to 110$, and $d \to 111$; $\bar{l} = 2$.

The average code word length is $\bar{\ell} = 1(.4) + 2(.3) + 3(.2) + 3(.1) = 1.9$.

The compression ratio is $\varphi \approx \frac{2}{1.9} \approx 1.05$.

The percentage of compression is $\rho \approx (1 - \frac{1.9}{2})100 = 5\%$.

The percentage of the remaining is $\delta \approx \frac{1.9}{2}100 = 95\%$.

Therefore, using this encoding scheme, an input sequence with a probability distribution $\mathcal{P}$ is expected to be compressed by 5%, or to generate an output sequence which is 95% of the size of the original input sequence. □

2.4 Static Coding

We now focus on the central problem of data compression. We have an input sequence, $\mathcal{X}$, which we want to transform into an output sequence, $\mathcal{Y}$, such that $\mathcal{Y}$ is expected to be shorter than $\mathcal{X}$. When $\mathcal{X}$ is completely recoverable from $\mathcal{Y}$ by a process called *de-*

compression, we are talking about *lossless data compression*. All the data compression methods discussed here are lossless.

The input sequence to be encoded is composed of symbols drawn from the source alphabet. These symbols have some statistical distribution. In *static coding*, the probabilities used to represent the distribution of the source are not modified in the entire encoding process. When these probabilities are changed (perhaps by a learning mechanism) during the encoding process, the process is called *adaptive coding*. The latter will be discussed later in this chapter.

Another classification that can be done in lossless compression is regarding to the information used to encode a given input sequence. Most of the compression methods used here use statistical information about the source. As mentioned earlier, this information is referred to as the probability of occurrence of the source alphabet symbols. All the methods discussed in this section are *statistical*, since they use statistical information about the source.

Most of the compression methods discussed here are based on a replacement encoding scheme. They are hence called *compression by a replacement scheme*. We have an input sequence to be encoded, we build an encoding scheme, and the process of encoding is achieved by replacing each of the source word symbols by a single code word according to the scheme. In this work, we assume that the encoding process is done by starting from the first (left most) symbol of $\mathcal{X}$, and continuing encoding each symbol until the last (right most) symbol is encoded. The decoding process is done in the same way.

Finally, all the methods discussed in this section are based on the zeroth-order modeling, i.e. the source is memoryless or zeroth-order.

2.4.1 Huffman's Algorithm

Huffman's algorithm was introduced by D.A. Huffman in 1952 [66]. It is a statistical method that generates an encoding scheme that satisfies the prefix condition. The idea of this method, as in the other statistical methods, is to assign shorter code words to symbols that occur with higher frequency, and conversely, assign longer code words to

those that occur less frequently. Huffman's algorithm is optimal, in the sense that it obtains an encoding scheme with the smallest average code word length.

This method proceeds by first generating a particular case of an *r-ary* tree, called *coding tree*, which is introduced in the following definition.

Definition 2.21. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. A *coding tree*, $\mathcal{T}$, is an r-ary tree with the following properties:

(*i*) Each node, q_i, is associated with a positive number, τ_i, called its *weight*.

(*ii*) The weight of an internal node is equal to the sum of the weights of its children.

(*iii*) Each leaf node is associated with either:

(a) a source alphabet symbol, in which case its weight is the probability of occurrence of this symbol, or

(b) a "null" symbol and its weight is 0.

Huffman's algorithm consists of two main steps. The first one involves the construction of a particular case of a coding tree, called the *Huffman tree*, defined in the following theorem. The second step is the construction of the prefix encoding scheme starting from this Huffman tree. A fundamental property of Huffman trees, referred to as the *sibling property*, is given in the following theorem, which is stated in terms of the binary code alphabet[3]

Theorem 2.5. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the code alphabet $\mathcal{A} = \{0, 1\}$. $\mathcal{T}$ is a Huffman tree if and only if its $2m - 1$ nodes can be arranged in the list: $q_1, q_2, \ldots, q_{2m-1}$ satisfying the following properties:

(*i*) $\tau_1 \geq \tau_2 \geq \ldots \geq \tau_{2m-1}$, and

(*ii*) for $j = 1, \ldots, m-1$, q_{2j} and q_{2j+1} are siblings in $\mathcal{T}$. □

[3]This property can also be generalized for multi-symbol code alphabets.

In [48], the properties of Theorem 2.5, are stated and proved for the binary code alphabet.

Let us start with the construction of the coding tree. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Prior to encoding, we need to consider the set $\mathcal{S}$ as a list sorted by decreasing order of probability of occurrence. We also need $m = r + k(r-1)$ for some positive integer, k, which can be achieved by adding symbols with probability 0 to the end of the list $\mathcal{S}$. The construction of the Huffman tree is summarized in Algorithm **Static_Huffman_Encoding**, in which τ_i is the weight of the node q_i.

Algorithm 1 Static_Huffman_Encoding

Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
Output: The Huffman tree, $\mathcal{T}$.
Method:
  for $i \leftarrow 1$ **to** m **do** *// Create a list of nodes with all the s_i's*
    Create a leaf node q_i with p_i and s_i
  endfor
  Create the list $\mathcal{Q}$ with all the nodes $\{q_i\}$
  while $|\mathcal{Q}| > r - 1$ **do** *// Construct the tree*
    Select the r smallest elements of $\mathcal{Q}$: $q_{i_1}, \ldots, q_{i_r}$
    Create a new node q_{new}
    $\tau_{new} \leftarrow \tau_{i_1} + \ldots + \tau_{i_r}$
    $q_{i_j} \leftarrow$ the jth children of q_{new}
    Remove $q_{i_1}, \ldots, q_{i_n}$ from $\mathcal{Q}$
    Insert q_{new} in $\mathcal{Q}$, such that $\mathcal{Q}$ keeps sorted
  endwhile
end Algorithm Static_Huffman_Encoding

After the Huffman tree is constructed, the encoding scheme must be generated. The tree $\mathcal{T}$ is annotated such that a branch leading to the jth child is labeled with a_j. The symbols s_i are arranged in the leaves.

The code word for the symbol s_i is derived by concatenating the labels encountered on the path from the root to the leaf associated with s_i.

Note that the labeling scheme is done by assigning a_j to the jth child from left to right. This labeling scheme is arbitrary. In fact, there are $r!$ different labeling alternatives in each node. Since $\mathcal{T}$ has $k+1$ internal nodes, by applying the multiplication rule

of combinatorics, $(r!)^{k+1}$ different encoding schemes can be generated with $\mathcal{T}$. In order to ensure consistency, an *a priori* convention between the encoder and the decoder must be maintained. They must both generate *exactly* the same tree and *exactly* the same encoding scheme from this tree by using the same labeling scheme.

Once the encoding scheme is generated, the encoding process is performed by replacing each input sequence symbol, $x[i]$, by its corresponding code word given by the encoding scheme. The output of the decoder is an output sequence, $\mathcal{Y}$, obtained by the concatenation of all the code words obtained by the replacement scheme.

The decoder proceeds by first generating a Huffman tree identical to that generated in the encoding phase[4]. With the tree generated, the decoding process is summarized in Algorithm **Static_Huffman_Decoding**, where $l_{\mathcal{Y}}$ represents the length of the output sequence, $\mathcal{Y}$. The convention used for labeling the nodes of the tree in building the encoding scheme consists of assigning a_j to the jth children.

Algorithm 2 Static_Huffman_Decoding

Input: The Huffman tree, $\mathcal{T}$. The output sequence, $\mathcal{Y}$. The code alphabet, $\mathcal{A}$.
Output: The original input sequence, $\mathcal{X}$.
Method:

```
  q ← root(T)
  k ← 1
  for j ← 1 to l_y do    // For every symbol of the input sequence
    i ← the index of y[j] in A
    q ← the i^th children of q
    if q is a "leaf" then   // A symbol of the source sequence is recovered
      x[k] ← the symbol associated with q
      q ← root(T)
      k ← k + 1
    endif
  endfor
end Algorithm Static_Huffman_Decoding
```

Example 2.8. Suppose the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$, and the binary code alphabet, $\mathcal{A} = \{0, 1\}$.

[4]The Huffman tree need not necessarily be constructed by the decoder again. It can instead be stored in the overhead of the compressed data.

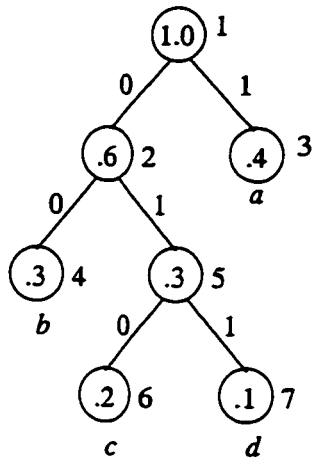


Figure 2.2: Huffman tree constructed with $\mathcal{S} = \{a, b, c, d\}$ with $\mathcal{P} = [.4, .3, .2, .1]$, and $\mathcal{A} = \{0, 1\}$.

The source alphabet is already sorted in decreasing order of frequency. The Huffman tree generated by using Algorithm **Static_Huffman_Encoding** is depicted in Figure 2.2. Observe that the tree satisfies the properties of Theorem 2.5. The integer number at the right of each node indicates the order in the list $\mathcal{Q}$ as in Algorithm **Static_Huffman_Encoding**.

Note that the nodes q_4 and q_5 can be interchanged resulting in a different Huffman tree. This decision is done after adding the weights of q_1 and q_2. The resulting node has weight .3, which has to be inserted in the list so that it is sorted after the insertion. Whether it is inserted before or after the node associated with b is based on the convention that the encoder and the decoder agree to.

Using the convention of assigning 0 to the left child and 1 to the right child, the resulting encoding scheme is $\phi : a \rightarrow 1,\ b \rightarrow 00,\ c \rightarrow 010,\ d \rightarrow 011$. The average code word length of the encoding scheme is $\bar{\ell} = 1.9$. □

2.4.2 Shannon's Method

Shannon's method was introduced in 1949 by C. Shannon [147]. This method is based on theoretical concepts of information theory, mainly on the noiseless coding theorem (Theorem 2.3). Although Shannon's method is less efficient than Huffman's algorithm

in terms of the compression ratio, it is faster, since each code word can be obtained directly from the probability of occurrence of the symbol.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Prior to encoding, $\mathcal{S}$ must be sorted, and rearranged if necessary, in non-increasing order of probability of occurrence. Besides, for each s_i, a cumulative probability function P_i is recursively calculated as follows:

$$P_i = \begin{cases} 0 & \text{if } i = 1 \\ P_{i-1} + p_i & \text{if } 2 \leq i \leq m\,. \end{cases} \tag{2.20}$$

The length, ℓ_i, for the code word w_i, is calculated as follows[5]:

$$\ell_i = \lceil \log p_i^{-1} \rceil\ . \tag{2.21}$$

Having calculated the length of w_i, P_i is converted from decimal to r-ary representation, resulting in an r-ary number like $0.w_{i_1}w_{i_2}\ldots$, where $w_{i_j} \in \mathcal{A}$, and w_i is the finite sequence $w_{i_1}w_{i_2}\ldots w_{i_{\ell_i}}$.

Note that Equation (2.21) can also be written as

$$\log p_i^{-1} \leq \ell_i < \log p_i^{-1} + 1\,. \tag{2.22}$$

From (2.22), we can see that Shannon's idea was really intuitive and directly based on the definition of the information content. Besides, it can be shown that the lengths calculated using Equation (2.22) satisfy Kraft's inequality, and hence lead to a prefix condition encoding scheme.

Example 2.9. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$.

[5] $\lceil x \rceil$ is the smallest integer n, such that $x \leq n$.

We have to calculate the cumulative probability for all the symbols of $\mathcal{S}$ using Equation (2.20), which result in $[0, .4, .7, .9]$.

The next step is to calculate the length of all the code words using Equation (2.21), which result in $\{2, 2, 3, 4\}$.

We now calculate the binary representation of each P_i. We have

$$\{0.000\ldots, 0.011\ldots, 0.1011\ldots, 0.11100\ldots\}.$$

Finally, the resulting encoding scheme is: $\phi : a \rightarrow 00$, $b \rightarrow 01$, $c \rightarrow 101$, and $d \rightarrow 1110$.

The average code word length of the encoding scheme is $\bar{\ell} = 2.4$. We observe that this average code word length is significantly higher than that obtained by Huffman's algorithm in Example 2.8, hence the compression ratio is smaller. Due to the poor efficiency (in regard to compression ratio), Shannon's method is not widely used in real life situations and became an interesting method for academic purposes only. □

2.4.3 Fano's Method

Fano's method was introduced by R. Fano [152]. Although this method is less efficient than Huffman coding, the compression ratio achieved by the former is very close to that of the latter. In Fano's method the coding tree is built in a top-down fashion, while in Huffman coding, the tree is constructed in a bottom-up fashion. We discuss Fano's method for the binary code alphabet.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \cdots, p_m]$, and the code alphabet $\mathcal{A} = \{0, 1\}$. Prior to encoding, the set $\mathcal{S}$ must be considered as a list and sorted in non-increasing order of probability of occurrence. The encoding process is summarized in Algorithm **Static_Fano_Encoding**. Although this algorithm produces only the encoding scheme, it can be easily adapted to output the coding tree as well. This is done by generating two new nodes each time a code alphabet symbol is pushed to the code word.

Let us discuss some observations about Fano's method. The list $\mathcal{S}$ is not optimally partitioned. Optimal partitioning is done in Huffman's algorithm, in which all the frequencies are considered each time that two symbols are merged. The problem of

Algorithm 3 Static_Fano_Encoding

Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
The code alphabet, $\mathcal{A}$.
Output: The encoding scheme, $\phi : s_i \rightarrow w_i$.
Method:

```
procedure Fano(S, P : list; A: set);
  Divide S and P into two sublists S_0, S_1 and P_0, P_1, such that the sum of probabilities of occurrence in each P_j is as nearly equal as possible.
  for j ← 0 to 1 do
    Push a_j to w_i for each s_i ∈ S_j
    if |S_j| > 1 then
      Fano(S_j, P_j, A)
    endif
  endfor
endprocedure
end Algorithm Static_Fano_Encoding
```

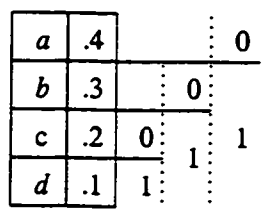


Figure 2.3: Fano encoding of the source $\mathcal{S} = \{a, b, c, d\}$ with frequencies $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$.

obtaining such a partitioning (as required by Fano's method) is NP-hard [152]. Another observation is about the strategy of assigning code alphabet symbols to the code words after each partition. As in Huffman's algorithm, we can choose among two different assignments at each stage (or when a new node in the coding tree is created). Therefore for $k+1$ internal nodes in the coding tree, we have $(2!)^{k+1}$ different encoding schemes.

Decoding in Fano's method is quite similar to Huffman's decoding. They only differ in the way that they construct the coding tree. In general, the coding tree built by Fano's method is not a Huffman tree. One of the cases in which the coding tree generated by Fano's method is a Huffman tree is when the probabilities of occurrence have the form $2^{-\ell_i}$.

Example 2.10. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with relative frequencies $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$.

The encoding process is shown in Figure 2.3. The dotted lines indicate that the list is divided in two sublists; 0 is assigned to the list on the top and 1 to the other list. The code word, w_i, is built by concatenating from right to left the code symbols of the sublists which s_i belongs to.

The resulting encoding scheme is: $\phi : a \to 0$, $b \to 10$, $c \to 110$, and $d \to 111$.

The average code word length of the encoding scheme is $\bar{\ell} = 1.9$. In this particular case, the average code word length is equal to that obtained by Huffman's algorithm in Example 2.8. □

2.4.4 Arithmetic Coding

The coding methods discussed above proceed by constructing an encoding scheme and then replacing each symbol of the input sequence by its corresponding code word. We refer to these methods as *encoding by replacement* schemes. These methods generate a prefix encoding scheme. In particular, Huffman coding minimizes the average code word length, hence it is optimal. However, there are some particular cases in which Huffman's algorithm can be sub-optimal.

Suppose that we have a long input sequence composed of symbols drawn from the source alphabet $\mathcal{S} = \{a, b\}$. Suppose also that the probabilities of occurrence of the symbols are $p_a = 0.95$ and $p_b = 0.05$. If we construct an encoding scheme with Huffman's algorithm, we obtain $\phi : a \to 0$ and $b \to 1$. The entropy of the source is approximately equal to 0.29, and the average code word length of the code is 1. This big difference is due to the fact that $\log p_a^{-1} \approx 0.074$, but we can not assign a code word with length less than 1 to the symbol a.

In order to solve this problem encountered by methods like Huffman's, P. Elias proposed an encoding method that replaces the entire input sequence by a code word (the output sequence) without building any encoding scheme [3]. This method, called *arithmetic coding*, is based on the following idea.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence

$\mathcal{P} = [p_1, \ldots, p_m]$. The input sequence $\mathcal{X} = x[1] \ldots x[M]$ is encoded as a real number in the interval [0,1). Unlike other statistical methods, ordering $\mathcal{S}$ by probability of occurrence is not required in arithmetic coding.

Prior to encoding the cumulative probability of each symbol is calculated using (2.20). We also need to set $P_{m+1} = 1$. A range must be assigned to each symbol s_i, given by the interval $[P_i, P_{i+1})$. Having done this, arithmetic coding proceeds as the procedure described in Algorithm **Static_Arithmetic_Encoding**, where the range produced by encoding $\mathcal{X}$ is given by $[\alpha, \beta)$, and the interval $[\alpha_k, \beta_k)$ corresponds to the sequence $x[1] \ldots x[k]$.

Algorithm 4 Static_Arithmetic_Encoding

Input: The input sequence, $\mathcal{X}$. The cumulative probabilities, $P_1, \ldots, P_m$.
Output: The range corresponding to the output sequence, $[\alpha, \beta)$.
Method:

```
α_0 ← 0    // Initialize [α, β) to be [0, 1)
β_0 ← 1
for k ← 1 to M do    // For every symbol of the source sequence
  i ← the index of x[k] in S
  α_k ← α_{k-1} + (β_{k-1} − α_{k-1}) P_i
  β_k ← α_{k-1} + (β_{k-1} − α_{k-1}) P_{i+1}
endfor
```

end Algorithm Static_Arithmetic_Encoding

Once the final range, $[\alpha, \beta)$, is obtained, we have to generate a code word from this range. There are some methods in order to do that. In [58], two methods are discussed. The main goal of these methods is to find the *dyadic fraction with least denominator* (*dfwld*).

In [145], a simpler method for creating the r-ary code word corresponding to the output sequence is given. First, we have to calculate a *tag*, $T(\mathcal{X})$, as follows:

$$T(\mathcal{X}) = \frac{\alpha + \beta}{2}. \tag{2.23}$$

The length of the output sequence is calculated as follows:

$$\ell_{\mathcal{Y}} = \left\lceil \log_r \frac{1}{\beta - \alpha} \right\rceil + 1 . \tag{2.24}$$

The output sequence, $\mathcal{Y}$, is then generated from the first $\ell_{\mathcal{Y}}$ symbols of the r-ary expansion of $T(\mathcal{X})$. In [145], it has been shown that the code obtained in this fashion is a uniquely decodable code.

The encoding technique discussed above is called the *pure* arithmetic coding scheme. The implementation of this method has some serious limitations. One of them is the fact that the interval is defined by real numbers. When long input sequences are encoded, the interval tends to be narrower, and with finite precision, α tends to be equal to β. In this situation, the decoder will not be able to recognize the interval in which $\mathcal{Y}$ falls. This is overcome by a technique called *rescaling* that proceeds by expanding the interval during the encoding process, except when the interval includes 0.5. A better solution for this problem is the use of integer arithmetic, in which case the interval includes integer numbers only. In the expansion, the particular case of the interval including 0.5 is well resolved by considering an *overflow/underflow* condition. The use of integer arithmetic for approximated dyadic fractions, scaling, underflow conditions, and bit stuffing can be found in [58, 65, 81, 82, 98, 145]. Other approaches proposed to improve the speed and compression efficiency of arithmetic coding can be found in [85, 158].

Decoding using the arithmetic method is achieved by generating the same ranges as those calculated by the encoder. Considering the pure arithmetic coding, the decoder proceeds as follows. The code word is converted into a real number, z, which lies in the interval $[0, 1)$. The first symbol, $x[1]$, is the symbol into whose range z falls. The new interval $[\alpha, \beta)$ is calculated and the intervals for all the symbols of $\mathcal{S}$ are recalculated. The process is continued until M source symbols are recovered. Clearly, the encoder has to send the source sequence length, M, to the decoder.

Example 2.11. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet $\mathcal{A} = \{0, 1\}$. Let us encode the input sequence $\mathcal{X} = adbc$.

k	$x[k]$	α_k	β_k
0		0	1
1	a	0	.4
2	d	.36	.4
3	b	.376	.388
4	c	.3844	.3868

Table 2.1: Arithmetic coding of the source word $\mathcal{X} = adbc$, where the source alphabet is $\mathcal{S} = \{a, b, c, d\}$ with probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$ and the code alphabet is $\mathcal{A} = \{0, 1\}$.

The ranges for the source symbols are: $[0, .4)$, $[.4, .7)$, $[.7, .9)$, and $[.9, 1)$.

The encoding process is shown in Table 2.1. The resulting interval, after encoding c at step $k = 4$, is $[.3844, .3868)$. Let us find the output sequence by first calculating $T(\mathcal{X})$ by using (2.23) as follows: $T(\mathcal{X}) = \frac{.3868+.3844}{2} = .3856$. The length of the output sequence is calculated by using (2.24) as follows: $\ell_{\mathcal{X}} = \lceil \log_2 \frac{1}{.3868-.3844} \rceil + 1 = 10$.

Finally, the binary expansion of .3856 is .**0110001010**1..., and hence the output sequence is $\mathcal{Y} = 0110001010$.

Assuming that the average source symbol length is $\bar{l} = 2$, the compression ratio, calculated by using Equation (2.17), is $\varphi = \frac{7}{10} = .7$. We expanded the original source word, instead of compressing it. Of course, this is an example in which we have one a, and when we have one c, whereas the expected number of a's is $4(.4) = 1.6$, and the expected number of c's is $4(.1) = .4$. □

It has been found that in most cases, arithmetic coding achieves better compression than Huffman's algorithm. There is some theoretical evidence in [58], which demonstrates that the average code word length achieved by arithmetic coding is very close to the entropy. In the case of the binary source alphabet the advantage of arithmetic coding is clear, it achieves compression whereas Huffman's algorithm does not.

2.4.5 Run Length Coding

Another important statistical method is *run length coding* [42, 43]. This method consists of intelligently replacing a run of a symbol by a counter. The length of the

counter and the compression ratio achieved is related to the probability distribution of the source symbols and the choice of the counter length.

For the binary source alphabet, the counter is encoded by choosing a k-digit binary number, where k has three different meanings:

(*i*) $k = 0$ means that the decoder will decode a single 1,

(*ii*) $0 < k < 2^k - 1$, the decoder will decode k 0's, and a 1, and

(*iii*) $k = 2^k - 1$, only k 0's are decoded.

The most important aspect in this method is the choice of k, which depends on the probability of 0, p_0. It can be optimized by minimizing the following equation as a function of k given the value of p_0 [57].

$$\frac{k}{1 - p_0^{2^k - 1}} \tag{2.25}$$

This method works very efficiently when p_0 is close to 1. The worst case is when p_0 is close to 0.5, since if p_0 is close to 0, the process can be optimized by negating the input sequence.

Some practical issues and performance improvements about this method are discussed in [63]. An extension of this method is presented in which $\mathcal{S}$ is considered to have characters, numbers, or spaces. A sequence of consecutive symbols in the input sequence is encoded by one or two symbols, a run length counter, a symbol, and a flag which indicates how the symbols and the counter are to be interpreted. This method is found to be highly effective when the input sequence has long runs of the same symbol, such as in databases or in the final stages of image and video compression [63, 114, 120].

2.5 Adaptive Coding

As discussed in section 2.4, static coding techniques work with some knowledge about the statistical distribution of the source. Suppose that the only information provided

to the encoder is the input sequence. The encoder can approximate the statistical distribution of the source by sampling the input sequence. However, there are practical situations in which this procedure may not be the most adequate solution. For example, when transmitting messages in real time between two points of communication, the receiver would not like to wait until the encoder produces a sequence of thousands of bits.

Adaptive methods solve this difficult situation by performing the encoding procedure and updating the statistical distribution at the same time. It seems to be difficult for the decoder to figure out what the encoder did and what distribution it used, since both are done at the same time. These issues are well managed by establishing a convention regarding the statistical distribution to start the encoding/decoding, and a convention as to what is the procedure to encode a source symbol and immediately update the statistical distribution of the source.

2.5.1 Huffman Coding

An adaptive approach of Huffman's algorithm was introduced by N. Faller in 1973 [45]. The basic idea of this approach is to maintain a Huffman tree, which is updated each time a symbol is encoded. Both the encoder and the decoder, start with the same tree, and perform the same updating of the tree during the encoding/decoding process.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. Prior to encoding, an initial tree, $\mathcal{T}_0$, is constructed by following the procedure described in Algorithm **Static_Huffman_Encoding**, and using a source alphabet with arbitrary probabilities of occurrence. $\mathcal{T}_0$ must be labeled, for example, by assigning '0' to each left child and '1' to each right child[6]. At the kth time step, the code word for $x[k]$ is generated by using the tree of the previous time step, $\mathcal{T}_{k-1}$. The code word is generated in the same way as it is done in static Huffman encoding, i.e. by annotating the tree from the root to the leaf associated with $x[k]$. After $x[k]$ is encoded, $\mathcal{T}_{k-1}$ has to be updated, generating $\mathcal{T}_k$ in such a way that the properties

[6] As discussed in Section 2.4.1, one of the 2^{k+1} labeling schemes can be selected. In order to ensure consistency, the same labeling scheme must be used in both the encoder and the decoder.

stated in Theorem 2.5 are satisfied. This process might take a long time, if a new Huffman tree is constructed each time a symbol is encoded.

Some attempts have been done in order to hasten the procedure for updating the tree. An efficient approach was introduced by R. Gallager [48]. Gallager's algorithm first updates the weight of the node visited and then interchange nodes, if necessary. Another approach, given in Algorithm **Adaptive_Huffman_Encoding**, was introduced by D. Knuth [75]. This algorithm is proposed for the binary code alphabet. In order to work correctly for an r-ary code alphabet, a special case must be considered because the last $r-2$ nodes in the list may have to be associated to a "null" symbol whose weight is 0.

Algorithm 5 Adaptive_Huffman_Encoding

Input: The Huffman tree after encoding $x[k-1]$, $\mathcal{T}_{k-1}$.
Output: The Huffman tree after encoding $x[k]$, $\mathcal{T}_k$.
Method:

```
q_i ← the node of T_{k-1} associated with x[k].
while q_i ≠ root(T_{k-1}) do
  q_j ← the left most node in the list whose weight is equal to τ_i.
  if q_i ≠ q_j then
    Interchange q_i and q_j.
  endif
  τ_i ← τ_i + 1
  q_i ← parent(q_i)
endwhile
T_k ← T_{k-1}
end Algorithm Adaptive_Huffman_Encoding
```

Decoding in adaptive Huffman coding also uses a Huffman tree. In order to ensure consistency, the tree has to be updated with exactly the same algorithm as that used by the encoder. The initial tree, $\mathcal{T}_0$, and the labeling scheme must also be exactly the same in both the encoder and the decoder. Given an input symbol, the decoder traces a path in the tree, starting from the root, and walking down in the direction of the leaves. Initially, a pointer is set to the root, and when a symbol is read, the pointer is set to the child that is labeled with this symbol. Once the pointer reaches a leaf, the symbol associated to that leaf is extracted and sent to the output so as to reconstruct

the original input sequence. The tree is then updated and the pointer is again set to the root.

2.5.2 Fano's Method

Fano's method can also be modeled using an adaptive approach. Although Fano's method is not widely used, little work has been done in order to improve its efficiency. To the best of our knowledge, there is no adaptive version of Fano's method proposed in the literature. We present two solutions to this problem in Chapters 4 and 5.

2.5.3 Shannon's Method

Shannon's method is the fastest among the statistical methods, but is the least desirable because it achieves the lowest compression performance. The adaptive version of Shannon's method can be obtained in the same way as Fano's method by maintaining a list sorted in non-increasing order of probability of occurrence, which is updated each time an input sequence symbol is encoded. The output sequence is constructed as it is done in the static version of Shannon's method. The frequencies are updated and this process is repeated for the next input sequence symbol. Although Shannon's algorithm seems to be faster than Fano's and Huffman's, its poor compression efficiency does not justify its implementation.

2.5.4 Arithmetic Coding

As in the case of the other statistical methods, arithmetic coding can also be rendered adaptive. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. Suppose that $\mathcal{P} = [p_1, \ldots, p_m]$ is composed of integer numbers such that each p_i is a counter that keeps track of the number of symbols s_i read so far from $\mathcal{X}$.

Prior to encoding, each p_i must be set to an arbitrary value. In order to ensure consistency, these values must be the same in both the encoder and the decoder. Adaptive arithmetic coding is summarized in Algorithm **Adaptive_Arithmetic_Encoding**.

Prior to encoding, the probabilities of occurrence (or counters) are all set to unity. In adaptive arithmetic coding the cumulative probabilities must be calculated at every time step. Although this calculation requires significant computation time, it can be optimized by recalculating only the ranges for s_i, $i > j$, where s_j is the last symbol encoded.

Algorithm 6 Adaptive_Arithmetic_Encoding

Input: The input sequence, $\mathcal{X}$.
Output: The range corresponding to the output sequence, $[\alpha, \beta)$.
Method:

```
for i ← 1 to m    // Initialize probabilities
  p_i ← 1/m
endfor
α_0 ← 0; β_0 ← 1
for k ← 1 to M    // For all the symbols of the input sequence
  P_1 ← 0
  for i ← 2 to m + 1 do    // Recalculate the cumulative probabilities
    P_i ← p_i / (k+m-1)
  endfor
  i ← the index of x[k] in S
  α_k ← α_{k-1} + (β_{k-1} − α_{k-1}) P_i
  β_k ← α_{k-1} + (β_{k-1} − α_{k-1}) P_{i+1}
  p_i ← p_i + 1
endfor
end Algorithm Adaptive_Arithmetic_Encoding
```

Decoding in arithmetic coding is similar to that of the static case, except that in the adaptive case, the probabilities of occurrence change at every time step, and they must be updated. In order to ensure consistency, both the encoder and the decoder must update these probabilities using the same procedure.

The adaptive arithmetic coding model discussed above is the so called "pure model". As in the static model, other advanced techniques can be applied to the pure adaptive arithmetic coding. These techniques include the use of integer arithmetic, the use of approximated dyadic fractions instead of real numbers for the probabilities, scaling, underflow conditions, and bit stuffing [58, 145].

2.5.5 Compression and Adaptation

The statistical adaptive methods discussed above, share some common procedural characteristics. One of them is that they simultaneously update the statistical distribution of the source and encode the input sequence. Therefore, we can separate the encoding procedure, which is particular to each method, from the probability updating process, which is similar in all the methods.

The information about the statistical distribution of the source tells us about the probability that a source symbol occurs in the input sequence, but does not say anything about the actual distribution of this symbol along the input sequence. For example, suppose that we have the input sequence $\mathcal{X} = a^k b^k$, where s_i^k means that s_i is repeated k times. The probabilities of occurrence of the source symbols are $p_a = p_b = .5$. If we encode $\mathcal{X}$ with a static method, the average code word length will be 1, achieving no compression. If we encode $\mathcal{X}$ with an adaptive method, the compression ratio can be increased by intelligently updating the statistical distribution of the source.

In the following paragraphs, we discuss some advanced techniques that update the statistical distribution of the source during the encoding process. These techniques have been successfully used in adaptive Huffman coding, and consequently they also require that changes be made in the Huffman tree, as soon as the statistical distribution of the source changes. All the approaches consider p_i to be a counter that keeps track of the number of times that s_i occurred so far. This counter can be an integer or a real number. The probability that a source symbol occurs in the input sequence is calculated as $\frac{p_i}{\sum_{i=1}^{m} p_i}$.

The first method, introduced by R. Gallager [48], consists of multiplying periodically each counter by a positive real number less than 1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$ which are counters of the source symbols in the input sequence after t symbols are encoded. All the counters, p_i's, are multiplied by a parameter α, where $\alpha \in (0, 1)$, and rounded down[7]. The value of t is calculated as follows:

[7]Gallager proposed p_i to be an integer number, but it can be a real number for which no rounding is necessary.

$$t = \left(\frac{1}{\alpha} - 1\right) m. \tag{2.26}$$

The adaptation of the statistical distribution of $\mathcal{S}$ to the remaining sequence to be encoded depends on the value of α. Gallager suggested that an appropriate choice is $\alpha = .5$, as it simplifies multiplications and keeps t relatively large.

The second method, introduced by G. Cormack and R. Horspool [30], suggests that the probabilities of occurrence should be real numbers instead of integers. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. They proposed exponential incrementing of the counters by choosing a multiplication factor $\alpha > 1$. At each time step, k, supposing that the input symbol is s_i, p_i is multiplied by α^k. Since α^k grows exponentially with k, and given the fact that we are using finite precision to represent the probabilities, sufficiently large values of α^k can cause overflow. In this case, we divide each p_i by α^k, and starting with $k = 1$, continue the encoding process as explained above.

As in Gallager's method, the choice of α determines whether the statistical distribution of the source is good or not for future encoding. The authors of [30] suggested a value of α slightly greater than unity, e.g. $\alpha = 1.01$.

The third method, discussed by T. Bell, J. Cleary, and I. Witten [13], consists of maintaining a window as a circular buffer, in which the last t symbols encoded are stored. Each time a new symbol, $x[k]$, is encoded, it is introduced in the window and its probability is incremented by one. The oldest symbol in the window is removed and its counter is decremented by one. In this method, the statistical distribution of the source depends on the last t symbols encoded. The quality of the statistical distribution of $\mathcal{S}$ depends on the size of the window, t, and the statistical distribution of the input sequence in the future.

2.5.6 Other Adaptive Methods

2.5.6.1 Interval Coding

A non-statistical adaptive method is the scheme known as *interval coding*, which was introduced by P. Elias [44]. This method operates with an arbitrary and infinite code word set, $\mathcal{C} = \{w_1, w_2, \ldots\}$, known by both the encoder and the decoder. This code satisfies the prefix property and $\ell_1 \leq \ell_2 \leq \ldots$, where ℓ_i is the length of w_i.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$.

Prior to encoding, $\mathcal{X}$ is expanded, generating a new source word $\mathcal{X}'$ obtained by concatenating the source alphabet symbols and $\mathcal{X}$ as follows: $\mathcal{X}' = x'[1] \ldots x'[M+m] = s_1 \ldots s_m x[1] \ldots x[M]$. The encoding is performed by replacing each source symbol, $x[k]$, by $w_j \in \mathcal{C}$, where j is the number of source word symbols between $x'[k+m]$ and its previous occurrence in $\mathcal{X}'$.

Decoding in interval coding is done by scanning the output sequence, $\mathcal{Y}$, from left to right. In the kth step, the code word $w_j \in \mathcal{C}$ is replaced by s_i, where $k - j$ is the position of s_i in $\mathcal{X}'$.

The compression efficiency of interval coding depends on the choice of $\mathcal{C}$ and the input sequence $\mathcal{X}$. Elias proposed two prefix codes that perform well for large source alphabets, but they perform worse than Huffman's algorithm. Since there is no technique for finding a specific code for a particular input sequence, and the codes proposed are not related to the statistical distribution of $\mathcal{S}$, interval coding is not an efficient method if evaluated in terms of its compression ratio.

2.5.6.2 Recency Rank Coding

Another non-statistical adaptive method is *recency rank coding*, which was also introduced by P. Elias [44]. Recency rank coding is quite similar to interval coding. The advantage of recency rank coding is that it uses a finite code word set, $\mathcal{C} = \{w_1, \ldots, w_m\}$. As in interval coding, $\mathcal{C}$ satisfies the prefix property and $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$, where ℓ_i is the length of w_i.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. As in interval coding, $\mathcal{X}$ is transformed into $\mathcal{X}'$ by concatenating $\mathcal{S}$ and $\mathcal{X}$ as follows: $\mathcal{X}' = x'[1] \ldots x'[M+m] = s_1 \ldots s_m x[1] \ldots x[M]$. The encoding is performed by replacing each source symbol, $x[k]$, by $w_j \in \mathcal{C}$, where j is the number of *distinct* source word symbols between $x'[k+m]$ and its previous occurrence in $\mathcal{X}'$.

The compression efficiency of recency rank coding depends on the choice of $\mathcal{C}$. Recency rank coding achieves a better compression ratio than interval coding, because it uses a finite code word set. In fact, encoding in the former is slower than in the latter: it is more difficult to count the number of distinct symbols in a block of symbols than to count the size of this block.

Although recency rank coding is included in this section, it is not a zeroth-order modeling method. As stated in Theorem 2.3, if we want lossless compression by a zeroth-order method, the entropy of the source must be less than or equal to the average code word length of any code. The following counter-example shows that recency rank coding is not zeroth-order.

Example 2.12. Consider the source alphabet $\mathcal{S} = \{a, b, c\}$ with probabilities of occurrence $\mathcal{P} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, and the code alphabet $\mathcal{A} = \{0, 1\}$. The entropy of the source is $\mathcal{H} \approx 1.58$.

Let us encode the input sequence $\mathcal{X} = ccccbbbbaaaa$ by recency rank encoding, using the code $\mathcal{C} = \{0, 10, 11\}$. The expanded input sequence is obtained by concatenating $\mathcal{S}$ and $\mathcal{X}$ as follows: $\mathcal{X}' = abcccccbbbbaaaa$. Replacing each $x[k]$ by the corresponding code word in $\mathcal{C}$, based on its last occurrence, is done as follows:

$x'[4] = x[1] = c$. The last occurrence of c in $\mathcal{X}'$ is $x'[3]$, then $w_1 = 0$ is sent to the output.

$x'[5] = x[2] = c$. Similarly, $x'[4] = c$, hence $w_1 = 0$ is sent to the output.

If we continue encoding in the same fashion, the resulting output sequence is: $\mathcal{Y} = 00001000011000$. The average code word length of $\mathcal{C}$, in this particular case, is $\bar{\ell} = \frac{14}{12} \approx 1.167 < 1.58 \approx \mathcal{H}$.

Since recency rank coding is a lossless method and there is a code whose average code word length is less than the entropy, recency rank coding is not zeroth-order,

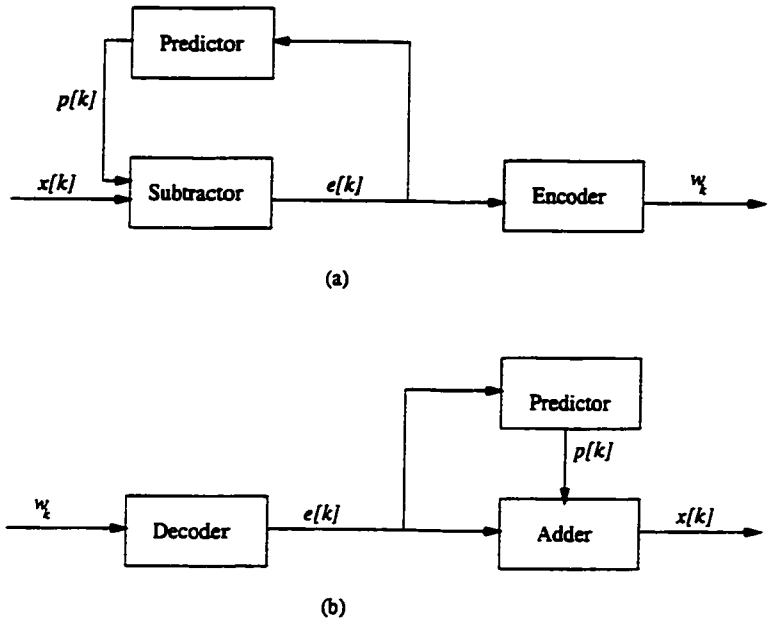


Figure 2.4: Predictive encoding and decoding.

according to the result of Theorem 2.3. □

2.5.6.3 Predictive Coding

The last method discussed in this section is *predictive coding*, which was introduced by P. Elias in 1955 [42, 43]. Predictive coding can be considered a statistical model, if it uses statistical information to make the predictions during the encoding process. It is also an adaptive method, because it can change the structure, statistics, and parameters during the encoding process.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. The encoder has a predictor (such as an algorithm, a function, or a mathematical formula), which is used to predict the next symbol, Figure 2.4(a). The output of the kth prediction, $p[k]$, is subtracted from $x[k]$, producing an error, $e[k]$. The value of $e[k]$ determines whether the prediction was good or not. The smaller the error, the better the prediction.

As the encoding process proceeds, a sequence of errors is generated. These errors can be encoded by using a pre-defined encoding scheme, assigning shorter code words

to smaller errors and, conversely, longer code words to larger errors. In fact, an encoding scheme can be cleverly constructed using any of the statistical methods discussed previously. In this case, the set of errors is the source alphabet and the relative frequencies of these errors can be calculated beforehand or adaptively during the encoding process. Observe in Figure 2.4 that the Encoder/Decoder is a separate module in the whole process.

In addition, the error is then introduced in the predictor so that the subsequent predictions are improved. Such "improvements" classify predictive coding as an adaptive technique.

In order to maintain consistency, the encoder and the decoder must have exactly the same predictor (see Figure 2.4(b)). For each code word that the decoder reads, it makes the predictions, and adds this prediction to the symbol read, producing the original input sequence symbol of $\mathcal{X}$. As in the encoder, the error is then introduced in the predictor so that the subsequent predictions are improved. Note that although predictive coding can perform both lossy or lossless compression [46, 51, 69, 92, 145], the contributions that we introduce in our study are intended for lossless compression.

2.6 Higher-order Coding

In Section 2.2.4, we discussed special kinds of sources called higher-order sources. In fact, when we speak about sources we refer to the way in which they are modeled. Given a particular source, whether or not the nature of the source is zeroth-order, higher-order, or any other non-statistical model, we intend to model it using a higher-order statistical model[8].

It is important to mention that, as in most compression techniques, statistical encoding methods perform by using certain probabilities which are either known *a priori* or are calculated adaptively during the encoding process. The nature of this process allows us to separate the statistical model from the coding method. Therefore, any of the statistical methods discussed earlier can be implemented using the higher-

[8]Such models are referred to as ***Markov models***, after the Russian mathematician Andrei Andrevich Markov (1856-1922) [145].

order model. In the interest of brevity, we present here only a short description of Huffman's method, and refer the interested reader to [58] for a description of other methods. We also discuss some other modeling techniques that model the source using different structures, but are essentially based on the higher-order models.

2.6.1 Huffman Coding

The idea of the higher-order Huffman coding is quite similar to the zeroth-order model. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. For the kth-order model, the conditional probability of s_i is $p(s_i|s_{i_1} \ldots s_{i_k})$. We are required to generate an encoding scheme for each context, and hence m^k different encoding schemes (or Huffman trees used in the decoding procedure) are to be generated.

Given an input sequence, $\mathcal{X} = x[1] \ldots x[M]$, the encoding procedure starts with an arbitrary context, and proceeds by encoding $x[j]$ using the encoding scheme corresponding to the context $x[j-k] \ldots x[j-1]$. The decoding procedure also knows the starting context used by the encoder, and proceeds in an identical fashion.

This procedure can also be rendered adaptive. After $x[j]$ is decoded, the conditional probabilities, and consequently, the encoding scheme (or Huffman tree) for the context $x[j-k] \ldots x[j-1]$ are updated.

Using the same idea, the Fano and arithmetic coding schemes can be designed to use the higher-order statistical model, either statically or adaptively.

Example 2.13. Consider the conditional probabilities, $\mathcal{P}^{(1)}$, given in Example 2.5. This is a first order model, and hence there are four contexts. We have four conditional probabilities for each context. Let us take those pertaining to the context a. The probabilities are given by the vector $\mathcal{P} = [.5, .25, .125, .125]$, which is the first row of the matrix $\mathcal{P}^{(1)}$. The resulting Huffman tree is depicted in Figure 2.5, and the encoding scheme is: $\phi : s_j \rightarrow w(s_j|a) : \{0, 10, 110, 111\}$. In a similar fashion, we can construct the Huffman trees and the encoding schemes for the other three contexts. □

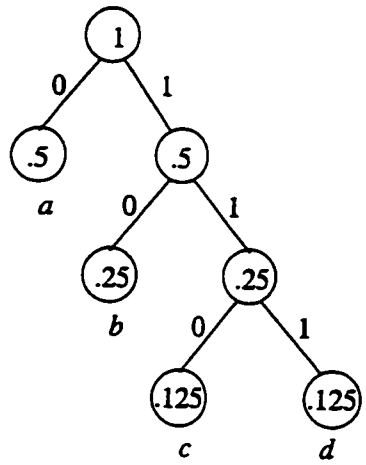


Figure 2.5: Huffman tree for the context a, constructed with $\mathcal{S} = \{a, b, c, d\}$, $\mathcal{A} = \{0, 1\}$, and $\mathcal{P} = [.5, .25, .125, .125]$ extracted from $\mathcal{P}^{(1)}$ given in Example 2.5.

2.6.2 Dictionary Based Coding

Data compression using a dictionary is an old idea based on the Braille code and its contractions [63]. The simplest approach works with a pre-loaded dictionary in which each subsequence is searched and replaced by an *index* or a *token*, which is generally shorter than the original sub-sequence. The main drawback is that the dictionary is static and the efficiency depends on the composition of the input sequence. We focus on more advanced techniques that use adaptive dictionary structures.

2.6.2.1 The LZ77 Compression Scheme

The first version of the LZ algorithms, LZ77, was introduced by Lempel and Ziv in 1977 [173]. The algorithm maintains a window or buffer that keeps a sequence of symbols previously encoded and the subsequent sequence to be encoded. The algorithm for radix $r > 1$ is discussed here.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, an input sequence $\mathcal{X} = x[1] \ldots x[M]$, and a buffer $B = b_1 \ldots b_{l_b - l_a} \ldots b_{l_b}$ that is composed of two parts: the first part, called the dictionary, contains the previously encoded output; the second part represents the *lookahead buffer*. A parsed sub-sequence, W, is maintained in the first part of the look ahead buffer, and another sub-sequence, W', that matches W is maintained in

the history buffer. Given as parameters to the algorithm are the size of B, l_b, and the maximum size of W, l_a. The history buffer contains $l_b - l_a$ symbols and the lookahead sequence contains up to l_a symbols.

The LZ77 algorithm divides $\mathcal{X}$ into $w_1 w_2 \ldots$ code words. Each w_k has a pre-defined length ℓ_{w_k} given by Equation (2.27), and is itself composed of three sub-words:

- w_{k_1}, the position of W' in B (encoded in radix r),
- $w_{k_2} = j$, the size of W (encoded in radix r), and
- $w_{k_3} = w_j$, the last symbol of W.

The size of the buffer and the size of the lookahead buffer are provided to the decoder. Given l_b and l_a, the length of w_{k_1} and w_{k_2} can be calculated by Equation (2.27). The length of w_{k_3} can be calculated as $\lceil \log_r m \rceil$, where r is the radix and m is the cardinality of $\mathcal{S}$. [9]

$$\ell_{w_k} = \lceil \log_r (l_b - l_a) \rceil + \lceil \log_r l_a \rceil + \lceil \log_r m \rceil . \tag{2.27}$$

At the start of the decoding process, the history buffer is initialized in the same way as it is done by the encoder. Each time the block w_k is read, a sub-sequence W is obtained by extracting w_{k_2} symbols from the history buffer starting from w_{k_1}, and concatenating the resulting string with w_{k_3}. Decoding in LZ77 is significantly faster than encoding since no matching tests are required.

One common problem with the LZ77 algorithm is that the last symbol of W is always included in the output, and so is not compressed. Moreover, this situation may be worse when the longest W matched in the history buffer has length 1. In this case, only one symbol is replaced by a code word, and this is generally longer than the symbol itself. This problem, among others, is solved with a modified version of the algorithm proposed by Storer and Szymanski [153]. Some authors refer to the latter algorithm as LZSS.

[9] In general, $m = 256$, i.e. the numbers of symbols of the ASCII code, and $r = 2$.

The basic LZ77 is not changed in LZSS. Rather a dramatic modification is made to the structure of w_k. Consider the binary code alphabet, where the first bit of w_k is a flag. If the flag is '1', it is followed by one symbol, s_i, encoded in $\lceil \log_2 m \rceil$ bits, which means that the longest W has only one symbol not found in the history buffer. In the other case, when W has more than one symbol, the flag value is '0', and is followed by an offset (the position of w'_1 in the history buffer) and the size of W.

Because of the efficiency of LZ77, many different implementations have been developed in order to improve the basic algorithm, from simple modifications in the structure of the dictionary, code word composition, to heavily parametrized models, such as QIC122 [63], LZRW1 (and other variations: LZRW1-A, LZRW2, LZRW3, LZRW3-A, LZRW4, and LZRW5) [58, 167, 168], which deal with various other extensions not discussed here in the interest of brevity.

2.6.2.2 The LZ78 Compression Scheme

Another dictionary method is the LZ78 algorithm, which was introduced in 1978 by Ziv and Lempel [174]. This technique is also called the Lempel-Ziv incremental parsing rule [143]. LZ78 introduces a dictionary, such that each new sub-sequence parsed in the input sequence is included in this dictionary. The longest sub-sequences parsed in the message that are matched with an entry in the dictionary are encoded by sending the *index*, or *token*, to the output. Good compression is achieved when long sub-sequences are found in the dictionary. The LZ78 algorithm is discussed here for $r > 1$, and a source alphabet of m symbols.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and an input sequence, $\mathcal{X} = x[1] \ldots x[M]$. LZ78 performs by searching each sub-sequence in a dictionary (pre-loaded with m entries: the symbols of $\mathcal{S}$). The index of this sub-sequence and the next character are sent to the output. The dictionary, is then updated by introducing this sub-sequence and the next character in the last position.

Decoding in LZ78 is significantly more straightforward. As in the encoder, the dictionary is loaded with m entries from $\mathcal{S}$. The encoded sequence is read by blocks of length $\lceil \log_r l_D \rceil$, where l_D is the length of the dictionary. Each block is converted

from radix r into an integer j. Then, the jth entry in the dictionary and the following symbol from the input is read and sent to the output. The dictionary is updated in the same way as it is done by the encoder. Decoding in LZ78 is faster than encoding, as the original sub-sequence is directly recovered from the dictionary, and consequently no testing for matching sub-sequences is necessary.

The most important extension of the LZ78 is the LZW algorithm introduced by T. Welch in 1984 [164]. Both algorithms are quite similar in the parsing rule, but they differ in the way in which the dictionary is updated. In LZW, the last sub-sequence not found in the dictionary is directly added as a new entry, whereas in LZ78, each time an output is produced, the dictionary is updated with m entries. Although LZW achieves similar compression to LZ78, the former is easier to implement. The asymptotic analysis of the efficiency of the LZ algorithms can be found in [143, 149].

Many extensions of the LZ78 and LZW algorithms, which solve some problems of these techniques, are the variant G of LZW [143], the LZC algorithm (the *compress* utility of Unix which has been replaced by *gzip*) [58], the BTLZ algorithm developed by British Telecom (the *V.42 bis* compression algorithm widely used in modems) [60], and the use of *flexible parsing*, called LZW-FP [93]. Other approaches based on LZ's algorithms can be found in [7, 12, 15, 50, 71, 83, 151, 163, 171].

2.6.3 Dynamic Markov Compression

Dynamic Markov Compression (DMC) was introduced by G. Cormack and R. Horspool in 1987 [31]. DMC is basically an implementation of the higher-order model using some structures to store the conditional probabilities.

The original idea of DMC was to implement a higher-order model for the binary source alphabet. Given an input sequence, $\mathcal{X} = x[1] \ldots x[M]$. DMC constructs a Finite State Machine (FSM) used to estimate the probability of the symbol being encoded at time n [18]. Each transition from state $\Phi[n]$ has associated with it a symbol, s_i, and a counter, $c_{s_i}^{\Phi[n]}$, which is used to calculate the conditional probability of occurrence of s_i given the current state, $\Phi[n]$. The conditional probability of s_i given $\Phi[n]$ is calculated as follows:

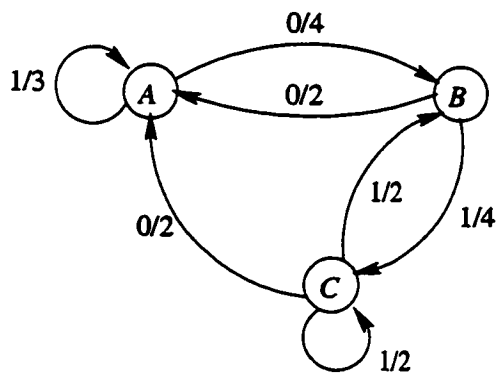


Figure 2.6: Example of an FSM representing an instance of the statistical information of a higher-order source using DMC.

$$p(s_i \mid \Phi[n]) = \frac{\kappa + c_{s_i}^{\Phi[n]}}{m\kappa + \sum_{j=1}^{m} c_{s_j}^{\Phi[n]}}, \tag{2.28}$$

where κ is a positive constant used to avoid zero probability values [31].

When encoding $x[n]$, the encoder considers the estimated conditional probability of $x[n]$ given the current state, $\Phi[n]$. The encoder generates the code word by using any of the statistical methods discussed above, such as Huffman, Arithmetic, or Fano. This makes use of the fact that, as noted earlier, we can separate the statistical model from the coding method. The encoder has also to update the counter, $c_{x[n]}^{\Phi[n]}$, and move to the corresponding next-state, $\Phi[n+1]$.

The structure of the model is also changed by a technique called *cloning*, which consists of duplicating the state Φ, generating a new state Φ'. As suggested in [31], cloning is done when there are two transitions (for s_i) from Φ_1 and Φ_2 to Φ, where $c_{s_i}^{\Phi_1}$ and $c_{s_i}^{\Phi_2}$ are greater than a pre-specified threshold[10]. After cloning, the transition (for s_i) from Φ_1 goes to Φ, and the transition from Φ_2 goes to Φ'.

Example 2.14. Consider the source alphabet $\mathcal{S} = \{0, 1\}$, and the code alphabet $\mathcal{A} = \{0, 1\}$. An example of an FSM representing a particular instant of time is depicted

[10]When there are more than two transitions (for s_i) to Φ, we check that the counter of s_i from $\Phi[n]$ to Φ is greater than a threshold and the counters from any other state for s_i to Φ are greater than another threshold [145].

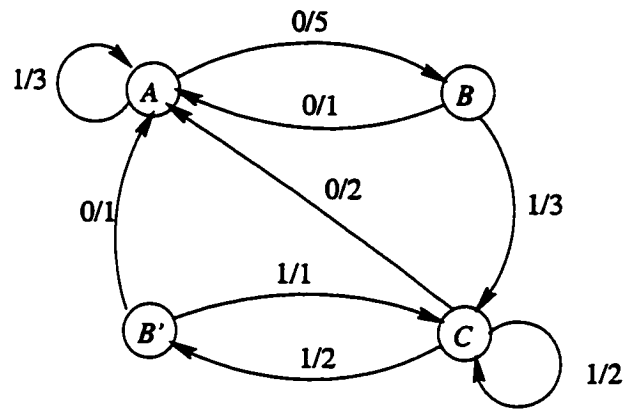


Figure 2.7: Finite State Machine representing the statistical information of the higher-order source given in Example 2.14.

in Figure 2.6.

Suppose that the current state is A, the threshold is 4, and a 0 comes from the input sequence. The probability of 0 given that the present state is A is given by $p(0 \mid A) = \frac{0.1+4}{2(0.1)+4+3} \approx 0.5694$, assuming that $\kappa = 0.1$. The counter of 1 from A to B has now to be changed to 5, which is greater than the threshold, and hence B must be cloned. The state B is duplicated, generating a new state B'. The new FSM of the model is depicted in Figure 2.7. □

It is important to remark that DMC is an easy-to-implement method, in the sense that only a few lines of code are required for the algorithm. It is also one of the best compression methods currently known, with regard to compression ratio [169]. One of the drawbacks of this method is that it requires a considerable amount of memory to store the FSM, which also has a direct consequence on the compression ratio that we want to optimize. Another aspect to consider in DMC is the statistical coding method to be used in encoding its predictions. The most widely used model is for the binary source alphabet for which arithmetic coding yields the best performance for a statistical coding method.

An extension of the DMC using a dictionary in each state of the FSM has been introduced in [159]. This approach has been reported to perform slightly worse in compression ratio but is faster than PPM described below.

2.6.4 Prediction with Partial Matching

Prediction with Partial Matching (PPM) is one of the most competitive compression methods used nowadays. It was introduced by J. Cleary and I. Witten in 1984 [28]. It can be seen that PPM is, in fact, a different way of implementing the higher-order model.

In Section 2.6, we stated that if we worked with a kth-order model, the number of possible conditional probabilities is m^{k+1}, where m is the number of symbols of the source. In real life data, most of the conditional probabilities are zero, and hence they need not be stored. In PPM, this problem is solved by storing only the non-zero probability transitions, using a sophisticated parsing of the input source by means of an "escape" symbol, and a variable size current context.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and an input sequence, $\mathcal{X} = x[1] \ldots x[M]$. At each encoding step, $x[n]$ is searched in the list of previously occurred symbols of the current context, $x[n - l_c] \ldots x[n - 1]$, whose size, l_c, is given as a parameter. If $x[n]$ is not found in the list of previously occurred symbols of the context $x[n - l_c] \ldots x[n - 1]$, it is added to it with a counter of unity, and an *escape* symbol is sent to the output. Then, $x[n]$ is searched in the list of previously occurred symbols of $x[n - l_c + 1] \ldots x[n - 1]$, continuing in this fashion until it is found. In this case, the counter of $x[n]$ in this context is used to estimate its conditional probability of occurrence given the current context, and the code word is then generated using a statistical coder. If $x[n]$ is not found in *any* of the contexts of length $l_c, l_c - 1, \ldots, 1$, it is encoded with probability $\frac{1}{m}$ (i.e. considering all the symbols of $\mathcal{S}$ equally likely).

One of the crucial problems in PPM is to calculate the probability of the *escape* symbol. Different approaches have been developed to solve this problem. A simple approach is to assign a counter of unity to the *escape* symbol. Another approach consists of using a counter c, where c is the number of different symbols that have occurred so far in the context [145]. A variant of this approach (called PPMC) was proposed in [97]. It also uses a counter, c, but calculates the probability of escape as $\frac{c}{n+c}$, where n is the total number of symbols seen so far in the current context. Another

variant, PPMD, calculates the probability of escape as $\frac{c}{2n}$ [64]. These variants provide marginal gain in the compression ratio.

Another problem in using the PPM is to determine the parameters of the model. Various attempts have been done in order to find the optimal ones, such as the use of a large context at the starting point of the encoding process, called PPMZ [27, 26], the Swiss Army Knife Data Compression (SAKDC) method [167], and a heavily parametrized model of PPM , including a family of estimators which combines a different model-selection technique and a hill-climbing mechanism [18, 19]. Other extensions of the PPM which improve its compression efficiency and speed can be found in [2, 41, 126].

2.6.5 Block Sorting Compression

The Block-sorting compression algorithm was first proposed by D. Wheeler and later published by M. Burrows and Wheeler in 1994 [20]. It is also known in the literature as the *Burrows-Wheeler Transform (BWT)*. Unlike other compression algorithms, BWT does not process the input sequentially. This algorithm performs a reversible transformation to the input sequence, producing a new sequence which is more "compressible" by statistical coders such as Huffman, Fano, or arithmetic coding.

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and an input sequence $\mathcal{X} = x[1] \ldots x[M]$. The first step of the BWT is to construct an $M \times M$ matrix, A, whose rows are the M permutations of $\mathcal{X}$ sorted *lexicographically*[11]. The output of the transformation is the last column of A, L, and, the index of the row in which $\mathcal{X}$ is located, I.

The inverse transformation is performed by sorting L lexicographically, generating a new list, F. The next step in decompression is to generate the list T, which establishes the correspondence between L and F. T is generated by scanning the symbols of F from left to right so that $T[i]$ is the position of $L[i]$ in F. The list T is used to reconstruct the original input sequence, $\mathcal{X}$. For each $i = 1, \ldots, M$, $x[M-i]$ is given by $L[T^i[I]]$, where $T^0[j] = I$, and $T^{i+1}[j] = T[T^i[j]]$.

[11]Since by our definition, the source alphabet is as a set, we use the convention that "lexicographically" means the following order: $s_1, s_2, \ldots, s_m$.

The gain in compression of BWT is given by the use of the Move-To-Front (MTF) algorithm to compress the sequence L. This algorithm is discussed in Section 2.5.6.2 and referred to as *recency rank coding*. We consider the source alphabet, $\mathcal{S}$, as a list, and scan L sequentially from left to right. A new sequence, $\mathcal{X}'$, is generated as follows: assign $j-1$ to $x'[i]$, where j is the position of $L[i]$ in $\mathcal{S}$. Then, move s_j to the front of $\mathcal{S}$ and continue in the same fashion with $L[i+1]$. The new input sequence, $\mathcal{X}'$, is encoded using any statistical coding method, such as Huffman, Fano, or arithmetic coding.

In practice, we expect quite good results by using the BWT for long sequences, as many zeroes are generated in $\mathcal{X}'$, and hence an arithmetic coding scheme is seen to be the most appropriate method. Although there is no theoretical evidence, the BWT and the PPM have been found to be one of the best compression methods – at least in some well-known standard benchmarks [169].

Some extensions have been proposed to improve the compression efficiency and speed of BWT and MTF algorithms. They can be found in [9, 11, 21, 40, 141, 171].

2.6.6 Grammar-Based Coding

A relatively new family of compression techniques are the Grammar-Based Coding (GBC) schemes. This method was first proposed in [29, 103, 153], and recently formalized in [72].

The basic idea of the GBC is the following. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, and a source sequence, $\mathcal{X} = x[1] \ldots x[M]$. After a transformation, a *context-free grammar*, G, is assigned to the source sequence, $\mathcal{X}$, such that $\mathcal{X}$ is the *language generated* by G. As in the theory of languages and compilers, G is a 4-tuple, $G = (V, T, P, S)$, where:

- V is the set of variables, or non-terminal symbols,
- T is the set of terminal symbols (in our case, $\mathcal{S}$),
- P is the set of production rules, and
- S is the start symbol.

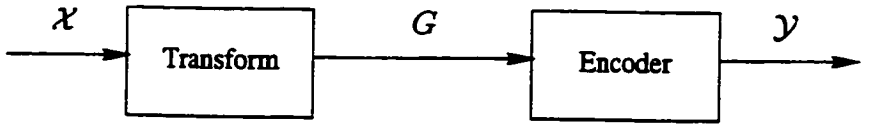


Figure 2.8: The Grammar-Based Coding transforming and encoding.

The GBC proceeds by generating an admissible, irreducible grammar, G, capable of generating $\mathcal{X}$ in the decoding process [72].

The compression process performed by the GBC is depicted in Figure 2.8, in which the basic stages of the GBC, transforming and encoding, are shown. The output sequence is the result of encoding G and the production rules necessary for the reproduction of the language generated by G, $\mathcal{X}$.

The GBC has been found to be more efficient (regarding compression ratio) than LZ77 and LZ78 for zeroth and higher-order randomly generated binary sources [170]. However, to our knowledge, no results have been reported which compare the power of the GBC with that of the well-known methods such as PPM, DMC, or BWT. A similar approach which uses grammars was also discussed in [80], and it has been reported to slightly outperform PPM on some particular non-standard benchmarks. More details of the formal theory, implementation, and empirical results of the GBC scheme can be found in [72, 170].

2.6.7 Other Higher-order Methods

Another higher-order modeling compression method is the Context Tree Weighting (CTW) method [165, 166], which is not discussed here since it is outside the scope of our study. Some experimental results on files of the Calgary Corpus and DNA sequences [142], seem to demonstrate that the CTW performs slightly better than PPM using the maximum order of five.

A hybrid model that merges some of the higher-order methods discussed earlier can be found in [160]. Other higher-order modeling techniques are the Universal Coding with Context Models [126], and the Compression via Multilevel Pattern Matching [73] schemes. These techniques, which are not discussed here, since they are outside the

scope of our study, have been reported to outperform the LZ family of algorithms.

2.7 Conclusions

In this chapter we presented an overview of lossless data compression. We discussed some of the formal concepts of information theory and coding theory, including the concepts of information content and entropy.

Using a statistical model for compressing data, the source model can be either a zeroth-order or a higher-order model. We have seen that statistical coding methods such as Huffman, Fano, or arithmetic coding can be implemented either by using a zeroth or a higher-order statistical model. The zeroth-order model is straightforward, but is less powerful than a higher-order model. However, the latter has some drawbacks primarily due to the amount of memory required, which grows exponentially with the order of the model.

Statistical methods can be implemented either statically or adaptively. In the static models, the statistical information of the source is not changed during the encoding process. Conversely, in the adaptive schemes, the encoding process and the estimation of the source probabilities are carried out simultaneously. The adaptive version of the Huffman and arithmetic coding methods were discussed in this chapter. To the best of our knowledge, the adaptive version of Fano's method has not been reported in the literature. We propose to study and develop the adaptive version of Fano's method later in our research endeavors.

In order to increase the compression ratio and simultaneously, solve the problem of implementing compression methods with a considerably higher-order statistical model, more sophisticated methods has been proposed in the literature. These methods were also discussed in the latter part of this chapter. Examples of these are the dictionary-based coding methods, LZ77 and LZ77, Prediction with Partial Matching (PPM) methods, Dynamic Markov Compression (DMC) methods, the Burrows-Wheeler Transform (BWT) methods, and the Grammar-Based Coding (GBC) schemes. All of them have been found to be very powerful in compressing files contained in the standard benchmarks. In particular, the BWT and PPM apparently achieve the best results when it concerns the compression ratio [169].

Chapter 3

Enhanced Fano Coding Schemes

One of the most well-known encoding methods is Huffman's algorithm. This method proceeds by generating the so called Huffman tree, by recursively merging symbols (nodes) into a new *conceptual* symbol which constitutes the internal node of the tree. In this way, Huffman's algorithm generates the tree in a bottom-up fashion. As opposed to this, Fano's method proceeds by generating a coding tree as well, but it proceeds in a top-down fashion. At each step the list of symbols is partitioned into two (or more, if the output alphabet is non-binary) new sublists, generating two or more new nodes in the corresponding coding tree. Although Fano's method, typically, generates a sub-optimal encoding scheme[1], the loss in compression ratio with respect to Huffman's algorithm can be relatively small, but also can be quite significant if the optimality of the partitioning is small.

The binary alphabet version of Fano's method proceeds by partitioning the list of symbols into two new sublists in such a way that the sums of probabilities of these two new sublists are as close to being equal as possible. This procedure is recursively applied to the new sublists until two atomic sublists with a single symbol are obtained, and simultaneously a bit is appended to the code words of each symbol in these sublists. As a result of this, if we start with the probabilities of occurrence in a decreasing order, it can be seen that, in general, symbols with higher probabilities can be assigned to longer code words than those with lower probabilities. This condition is not desirable;

[1]This is due to the fact that the optimal partitioning in Fano's method is NP-hard [152].

we attempt to rectify this condition in Fano$^+$, a superior Fano-based scheme which we develop in this chapter.

Since this does not happen in the family of *optimal* encoding algorithms (such as Huffman's), in this chapter, we introduce a front-end process to the static Fano encoding scheme, which achieves a code word rearrangement in such a way that the code word lengths are sorted in an increasing order of lengths. Indeed, we utilize the result of an archaic theorem concerning rearrangements of two sets [59] which, in our case, are the probabilities of the symbols and the lengths of the code words to yield a superior and enhanced performance.

On the other hand, after constructing a coding tree (such as a Huffman tree or a Fano tree), an encoding scheme is generated from that tree by labeling the branches with the code alphabet (typically, binary) symbols. Given a tree constructed from a source alphabet of m symbols, 2^{m-1} different encoding schemes can be generated. Of these, only one of them is of a family of codes known as *canonical codes*, which are defined as follows [61, 169].

Definition 3.1. Let $\mathcal{A} = \{0, 1\}$ be the code alphabet, and $\phi = \{w_1, \ldots, w_m\}$ be a code, where ℓ_i is the length of w_i, and $w_i \in \mathcal{A}^+$. ϕ is a *canonical code* if the following conditions are satisfied for $i, j = 1, \ldots, m$, $i < j$:

(*i*) $\ell_i \leq \ell_j$, and

(*ii*) $w_i \prec w_j$, where "$\prec$" stands for the relation "lexicographically precedes" in a standard dictionary whose alphabet is $\mathcal{A}$.

For example, for the binary code alphabet, $\phi = \{00, 01, 100, 101, 110, 111\}$ is a canonical code. Observe that $00 \prec 01 \prec 100 \prec 101 \prec 110 \prec 111$, and that the code words are arranged in an increasing order of length, namely $\{2, 2, 3, 3, 3, 3\}$.

Canonical codes are desired because they allow *extremely* fast decoding, and require approximately *half* of the space used by a decoding tree. Obtaining canonical codes using Huffman's algorithm is not straightforward. In fact, it requires the execution of very complicated mechanisms, involving a three-phase process for calculating the code word lengths, and then invoking an extra process for obtaining the canonical

code itself. Although Huffman's algorithm and Fano's method both take $O(m \log m)$ time[2], constructing the canonical code using the latter, is a straightforward matter. As mentioned above, Fano's method generates a coding tree in a top-down fashion. During the tree construction, the canonical code is obtained by assigning '0' to the left branches and '1' to right branches. Moreover, rearranging the code word symbols in an increasing order of length (the enhancement introduced here) can be done by rearranging the corresponding source alphabet symbols, again resulting in a canonical code.

In this chapter, we introduce Fano$^+$, an enhanced version of the static Fano coding approach utilizing the concept which we called "code word arrangement", and which is based on a fundamental property of two lists (the code words in terms of their lengths and probabilities) arranged in an increasing and decreasing order respectively [59]. This chapter details the encoding algorithm, the decoding algorithm, and the partitioning procedures suitable for Fano$^+$, and a rigorous analysis of their respective properties.

We finally discuss some empirical results obtained from running static Huffman's algorithm, the static version of the traditional Fano coding, and Fano$^+$ on real life data.

3.1 The Enhanced Coding Algorithms

3.1.1 Motivation

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq p_2 \geq \ldots \geq p_m$. Unless otherwise stated, in this chapter, we assume that the code alphabet is $\mathcal{A} = \{0, 1\}$. One of the properties of the encoding schemes generated by Huffman's algorithm is that $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$. In general, this property is not satisfied by the encoding schemes generated by Fano's method.

It is also important to mention that if the code word lengths are not in this order, the coding tree does not satisfy the sibling property. Before we proceed with the

[2]The most efficient mechanism for generating the canonical code has been found to take $O(k + k \log \frac{m}{k})$ time and space, where k is the number of different probabilities [99], and hence, in the worst case, it takes $O(m \log m)$.

theoretical analysis of the relation between the code word lengths and the sibling property, we introduce the following recursive definition.

Definition 3.2. Consider a coding tree $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ in which each leaf is associated with a source alphabet symbol, s_i. The code word length of a node t_j, where t_j is any node of the tree, is defined as follows:

(*i*) 1 if t_j is a leaf, or

(*ii*) $1 + \text{length}(t_{j_c})$, where t_{j_c} is the left or right child of t_j, whenever t_j is an internal node.

In terms of notation, as mentioned in Chapter 2, we refer to the weight of the node t_i as τ_i, where $\tau_i = \sum_{s_j \in \mathcal{T}_i} p_j$, and $\mathcal{T}_i$ is the subtree rooted at node t_i.

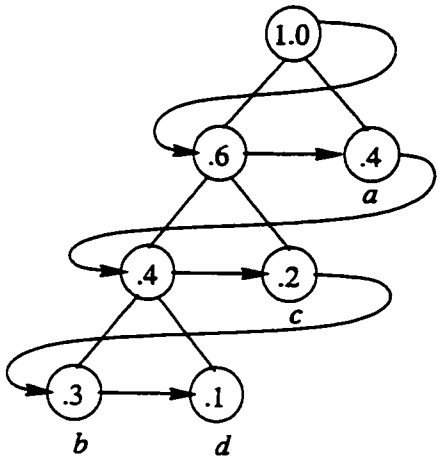
Using this definition, we shall prove that whenever the probabilities of occurrences of the input symbols are in a non-increasing order, if the code words are not in an increasing order of length, the coding tree associated with such an encoding scheme does not satisfy the sibling property. This fact is clarified in Example 3.1 below, and formally stated and proved in Theorem 3.1. It is thereafter used in Fano$^+$ to modify the coding tree so as to get an enhanced performance.

Example 3.1. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ whose probabilities of occurrence are $\mathcal{P} = [0.4, 0.3, 0.2, 0.1]$. Suppose that a particular coding tree, $\mathcal{T}$, constructed from $\mathcal{S}$ and $\mathcal{P}$ is the one depicted in Figure 3.1.

The complete table of the indices, the probabilities of occurrence, $\mathcal{P}$, and the code word lengths, $\{\ell_i\}$, for the symbols are given in the following table:

s_i	p_i	ℓ_i
a	0.4	1
b	0.3	3
c	0.2	2
d	0.1	3

Observe that $\ell_i = 3 > 2 = \ell_j$, where $i = 2 < 3 = j$. The corresponding locations in the list are t_5 and t_6 whose weights are 0.2 and 0.5 respectively. Clearly, the sibling



List Id	τ_i
t_1	1.0
t_2	.6
t_3	.4
t_4	.4
t_5	.2
t_6	.3
t_7	.1

t_{k_i}	k_i
t_{k_1}	3
t_{k_2}	6
t_{k_3}	5
t_{k_4}	7

Figure 3.1: An example of a coding tree that does not satisfy the sibling property. Note that the list positions we referred to as t_i, and the position of the symbols of $\mathcal{S}$ in the list are t_{k_i}.

property is violated. Since $0.3 = p_2 > p_3 = 0.2$, Condition (ii) of the sibling property is not satisfied. □

Example 3.1 depicts a rather simple case, which is not going to occur when constructing the tree for this scenario using Fano's method. However, when the number of symbols is large and the probabilities of occurrence are not skewed, there are some cases in which the coding tree constructed using Fano's method is similar to the one shown in Theorem 3.1.

Theorem 3.1. Let $\mathcal{T}$ be a coding tree constructed from the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$ are sorted, i.e. $p_1 \geq \ldots \geq p_m$. For any encoding scheme, $\phi : \mathcal{S} \rightarrow \{w_1, \ldots, w_m\}$, obtained after labeling as per the tree $\mathcal{T}$, where ℓ_i is the length of w_i, if there exist i and j such that $i < j$ and $\ell_i > \ell_j$, then $\mathcal{T}$ does not satisfy the sibling property.

Proof. Let $\mathcal{T}^\star = \{t_1^\star, \ldots, t_{2m-1}^\star\}$ be a coding tree. $\mathcal{T}^\star$ satisfies the sibling property if and only if its nodes can be arranged in a sequence, $t_1^\star, t_2^\star, \ldots, t_{2m-1}^\star$, such that for any non-root node, the following conditions are satisfied:

(*i*) t^*_{2i} and t^*_{2i+1} are siblings in $\mathcal{T}^*$, $i = 1, \ldots, m-1$, and

(*ii*) $\tau^*_2 \geq \tau^*_3 \geq \ldots \geq \tau^*_{2m-1}$.

Let $\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$, where ℓ_i is the length of w_i, be an encoding scheme generated from the coding tree $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. Suppose that $\exists$ i and j, $i < j$, such that $\ell_i > \ell_j$.

We know that every symbol of $\mathcal{S}$ is associated with a leaf of $\mathcal{T}$ and vice versa. Without loss of generality, suppose that s_i is associated with t_k, and s_j is associated with t_l. We consider three cases for the leaves, t_k and t_l.

(*i*) t_k and t_l are siblings. Using Definition 3.2, we see that ℓ_i should be equal to ℓ_j, and hence we need not consider this case, as it is excluded by virtue of the fact that we consider code words for which $\ell_i > \ell_j$ in this theorem.

(*ii*) t_k and t_l are not siblings, but they are in the same level. Again, from Definition 3.2, ℓ_i should be equal to ℓ_j. This implies that this case can also be excluded by virtue of the fact that we consider code words for which $\ell_i > \ell_j$ in this theorem.

(*iii*) t_k and t_l are not in the same level (obviously, they are not siblings). This leads us to four distinct cases for t_k and t_l enumerated below:

(a) $t_k = t_{2u}$ and $t_l = t_{2v}$ for some u and v, and both are the left children of a node, where ℓ_i is the depth of t_k and ℓ_j is the depth of t_k. Since $\ell_i > \ell_j$, the level in which t_k is located is greater than that of t_k. This implies that if we arrange the nodes from left to right so as to satisfy Condition (*ii*) of the sibling property, we obtain the following list:

$$t_1, \ldots, t_l, \ldots, t_k, \ldots, t_{2m-1} .$$

But since $p_i \geq p_j$, $\tau_k \geq \tau_l$, which violates Condition (*ii*) of the sibling property. The result follows.

The three remaining cases are proved in an analogous way. In order to avoid repetition, we only list them.

(b) $t_k = t_{2u}$ and $t_l = t_{2v+1}$, in which case t_k is a left child and t_l is a right child.

(c) $t_k = t_{2u+1}$ and $t_l = t_{2v}$, where t_k is a right child and t_l is a left child.

(d) $t_k = t_{2u+1}$ and $t_l = t_{2v} + 1$, which implies that both t_k and t_l are right children.

In each of this cases, as in Case (a), it can be seen that the node t_k precedes t_l in the list, and hence the sibling condition is violated.

The result follows. □

The proof of Theorem 3.1 illustrates the fact that when there is a leaf whose weight is greater than that of a leaf located in an upper level of the tree, the sibling property is not satisfied. As a consequence, the average code word length of any encoding scheme obtained from this tree is not minimal.

3.1.2 The Enhanced Coding Algorithms

Considering this fact, we now propose Fano$^+$ using the following modification to the traditional static Fano's method. It is well known that Fano's method requires that the source symbols and their probabilities are sorted in a non-increasing order of the probabilities. What we incorporate is as follows: After all the code words are generated, we sort them in terms of their increasing order of lengths maintaining $\mathcal{S}$ and $\mathcal{P}$ in the order of the probabilities. This enhancement leads to Fano$^+$, a modified Fano coding algorithm which generates encoding schemes whose average code word lengths are arbitrarily close to those of the optimal ones (i.e. those generated by Huffman's algorithm). This enhancement is formalized in Rule 3.1 given below.

Rule 3.1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq p_2 \geq \ldots \geq p_m$. Suppose that

$\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$

is the encoding scheme obtained by Fano's method.

Rearrange $w_1, \ldots, w_m$ into $w'_1, \ldots, w'_m$ such that $\ell'_i \leq \ell'_j$ for all $i < j$, and maintain $s_1, \ldots, s_m$ in the same order, obtaining the following encoding scheme :

$\phi' : s_1 \rightarrow w'_1, \ldots, s_m \rightarrow w'_m$. □

The encoder constructs a code, ϕ, using the static Fano's method, and then applying Rule 3.1, obtains the enhanced encoding scheme ϕ'. The decoder invokes the same rule obtaining ϕ', from which it generates the decoding scheme, $(\phi')^{-1} : w'_1 \rightarrow s_1, \ldots, w'_m \rightarrow s_m$.

In Example 3.1, swapping the symbols b and c (and updating the content of the tree), is equivalent to swapping their corresponding code words in any of the encoding schemes generated from $\mathcal{T}$. Observe that after this modification, $\mathcal{T}$ satisfies the sibling property. Note that we do not obtain the optimal encoding algorithm, like Huffman's, since we are only swapping code words, and hence the coding tree is the same as that constructed by Fano's method. There are some cases in which the structure of the tree must be changed so that it satisfies the sibling property. In the interest of completeness, the formal Algorithm **Enhanced_Fano_Encoding** and Algorithm **Enhanced_Fano_Decoding** are given below. Observe that unlike the encoding algorithm which does not explicitly create a Fano tree, to efficiently achieve decoding, the decoding algorithm also maintains this Fano tree. This is the version we give below, and leave the implementation details to the interested reader.

3.2 Properties of the Enhanced Fano Coding

To facilitate the analysis, we first introduce two important properties of Fano$^+$, the enhanced static Fano coding. The first relates to the efficiency in compression achieved by Fano$^+$, and the second is the property that it achieves lossless compression.

The efficiency of compression of Fano$^+$ is a direct consequence of the rearrangement of the code words such that they are sorted in an increasing order of length (Rule 3.1). This is stated in the following theorem, for which a proof[3] can be found in [59].

[3]The proof given in [59] is also valid for all real, not necessarily positive $p_1, \ldots, p_m$ and $\ell_1, \ldots, \ell_m$.

Algorithm 7 Enhanced_Fano_Encoding

Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
The code alphabet, $\mathcal{A} = \{0, 1\}$. The source sequence, $\mathcal{X} = x[1] \ldots x[M]$.
Output: The output sequence, $\mathcal{Y} = y[1] \ldots y[R]$.
Method:

```
procedure Fano(S, P, A : list; var {w_i} : code)
  Divide S and P into two sublists S_0, S_1 and P_0, P_1, such that the sum of probabilities of occurrence in each P_j is as nearly equal as possible.
  for j ← 0 to 1 do
    Push j to w_i for each s_i ∈ S_j.
    if |S_j| > 1 then
      Fano(S_j, P_j, A, {w_i})
    endif
  endfor
endprocedure
Fano(S, P, A, {w_i})
{w_1, ..., w_m} ← {w'_1, ..., w'_m}, where ℓ'_1 ≤ ... ≤ ℓ'_m
k ← 1
for i ← 1 to M do
  y[k] ... y[k + ℓ_j − 1] ← w_j, where x[i] = s_j
  k ← k + ℓ_j
endfor
end Algorithm Enhanced_Fano_Encoding
```

Algorithm 8 Enhanced_Fano_Decoding

Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
The code alphabet, $\mathcal{A} = \{0, 1\}$. The encoded sequence, $\mathcal{Y} = y[1] \ldots y[R]$.
Output: The original source sequence, $\mathcal{X} = x[1] \ldots x[M]$.
Method:

```
procedure Fano(S, P, A : list; n : node);
  Divide S and P into two sublists S_0, S_1 and P_0, P_1, such that the sum of proba-
    bilities of occurrence in each P_j is as nearly equal as possible.
  Create two new nodes: n_0 and n_1
  L_n ← n_0; R_n ← n_1
  for j ← 0 to 1 do
    τ_{n_j} ← Σ_{p_i ∈ P_j} p_i
    if |S_j| > 1 then
      Fano(S_j, P_j, A, n_j)
    else
      σ_{n_j} ← S_j   // The symbol associated with the leaf n_j
    endif
  endfor
endprocedure
Create a new node: root
τ_root ← Σ_{i=1}^{m} p_i
Fano(S, P, A, root)
∀i and j, such that p_i > p_j, swap s_i and s_j in the tree if ℓ_i > ℓ_j .
k ← 1; n ← root
for i ← 1 to R do
  if y[i] = 0 then
    n ← L_n
  else
    n ← R_n
  endif
  if n ='leaf' then
    x[k] ← σ_n; k ← k + 1   // σ_n is the symbol associated with node n
  endif
endfor
end Algorithm Enhanced_Fano_Decoding
```

Result 3.1. Let $p_1, \ldots, p_m$ be positive real numbers such that $p_1 \geq \ldots \geq p_m$, and let $\ell_1, \ldots, \ell_m$ be positive integers such that $\ell_1 \leq \ldots \leq \ell_m$. Then, for any rearrangement $\ell'_1, \ldots, \ell'_m$ of the list $\ell_1, \ldots, \ell_m$,

$$\sum_{i=1}^{m} p_i \ell_i \leq \sum_{i=1}^{m} p_i \ell'_i . \tag{3.1}$$

Observe that from Result 3.1, we can infer that for any list of code word lengths, we can obtain the optimal rearrangement, i.e. the one that satisfies $\ell_1 \leq \ldots \leq \ell_m$, by using Rule 3.1. However, this does not guarantee that we obtain the optimal encoding scheme (as in Huffman's algorithm), which also depends on constructing the optimal coding tree for a particular set of probabilities, $p_1, \ldots, p_m$.

The following counter-example shows that even the enhanced Fano coding algorithm does not ensure the optimal encoding scheme.

Example 3.2. Consider the source alphabet $\mathcal{S} = \{a, b, c, d, e\}$ whose probabilities of occurrence are $\mathcal{P} = [.35, .17, .17, .16, .15]$.

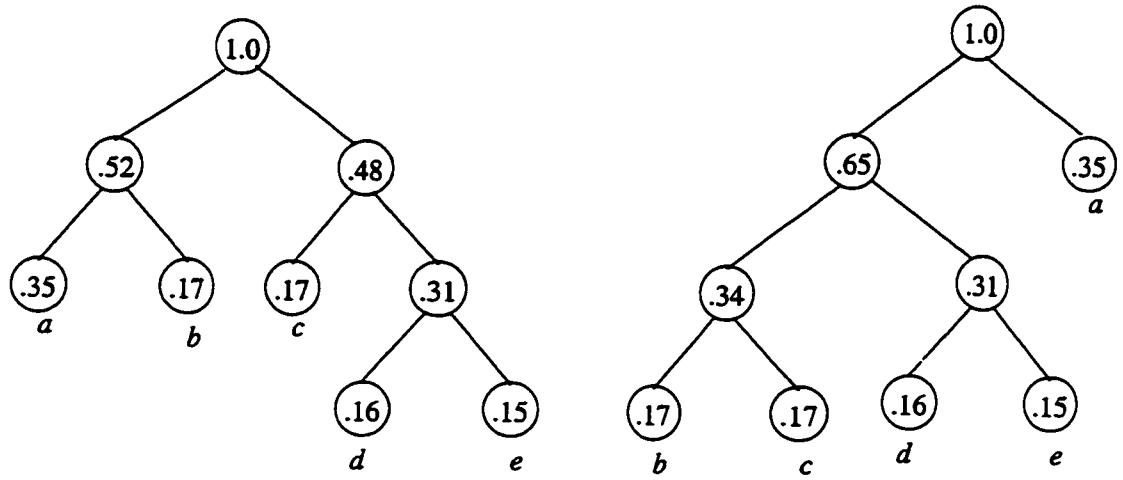
The coding tree constructed using the conventional Fano's method is depicted in Figure 3.2 (a). The corresponding Huffman tree is depicted in Figure 3.2 (b).

Observe that even after rearranging the code words obtained from labeling the tree of Figure 3.2 (a), it is not possible to achieve the optimal encoding scheme. This fact will also be observed in the empirical results shown later in this chapter. □

Prior to analyzing the correctness of Fano$^+$, and showing that it achieves lossless compression, we prove a result that will be needed later. This result is stated in Theorem 3.2, in which we prove that the enhanced static Fano's method generates a prefix code.

Theorem 3.2. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet, and $\mathcal{A} = \{0, 1\}$ be the binary code alphabet. Procedure Fano(...) of Algorithm **Enhanced_Fano_Encoding** generates a prefix code.

Proof. We prove this theorem by an induction on the number of steps used in the partitioning.



(a) Using Fano's method.

(b) Using Huffman's algorithm.

Figure 3.2: Two different coding trees constructed with the source alphabet and probabilities of occurrence given in Example 3.2.

Let $\mathcal{S}(j)$ be the list being partitioned at time 'j', i.e. the jth partitioning step of procedure Fano(...). At the time instant 'j', $\mathcal{S}(j)$ is partitioned into $\mathcal{S}_0(j)$ and $\mathcal{S}_1(j)$. The symbol '0' is added as a *suffix* to all the code words of $\mathcal{S}_0(j)$, and '1' is added as a *suffix* to all the codewords of $\mathcal{S}_1(j)$.

The basis case is straightforward, as it involves the first partitioning in which $\mathcal{S}(1)$ is partitioned into $\mathcal{S}_0(1)$ and $\mathcal{S}_1(1)$, where the code words of $\mathcal{S}_0(1)$ start with '0' and the code words of $\mathcal{S}_1(1)$ start with '1'. Clearly, no code word of $\mathcal{S}_0(1)$ is a prefix of any code word in $\mathcal{S}_1(1)$ and vice versa.

We now proceed to the inductive step in which the hypothesis is that no code word in $\mathcal{S}_0(j-1)$ is a prefix of *any* code word in $\mathcal{S}_1(j-1)$ and vice versa.

We have three mutually exclusive and exhaustive cases, which are the following:

(*i*) $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}\}$, where both $\mathcal{S}_0(j)$ and $\mathcal{S}_1(j)$ have the cardinality unity.

(*ii*) $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}, \ldots, s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$.

(*iii*) $\mathcal{S}_0(j) = \{s_i, \ldots, s_{i+v-1}\}$ and $\mathcal{S}_1(j) = \{s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$.

We consider these three cases separately.

(*i*) In this case, the two sublists resulting from the partitioning of $\mathcal{S}(j)$ are $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}\}$.

First of all, from the inductive hypothesis, no code word in $\mathcal{S}_0(j-1)$ is a prefix of any code word of $\mathcal{S}_1(j-1)$, and vice versa. We now have to deal with $\mathcal{S}(j)$ which is either $\mathcal{S}_0(j-1)$ or $\mathcal{S}_1(j-1)$. Since '0' is added as a *suffix* to w_i (the code word corresponding to s_i), and '1' is added as a *suffix* to w_{i+1} (the code word corresponding to s_{i+1}), it necessarily means that w_i has the form $w_{i_1} \ldots w_{i_n}0$ and w_{i+1} has the form $w_{i_1} \ldots w_{i_n}1$, where $w_{i_1}, \ldots, w_{i_n} \in \mathcal{A}$. This ensures that neither w_i nor w_{i+1} are prefixes of each other. This completes this case.

(*ii*) In this case, the two sublists resulting from the partitioning of $\mathcal{S}(j)$ are of the form $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}, \ldots, s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$. As mentioned above, as a consequence of the inductive hypothesis, no code word of $\mathcal{S}(j)$ is a prefix of any code words that are not in $\mathcal{S}(j)$, and vice versa.

In addition, after '0' is added as a *suffix* to w_i (the code word corresponding to s_i) and '1' is added as a *suffix* to $w_{i+1}, \ldots, w_{i+v}$ (the code words corresponding to $s_{i+1}, \ldots, s_{i+v}$), w_i has the form $w_{i_1} \ldots w_{i_n}0$, and all the code words in $\mathcal{S}(j)$, $w_{i+1}, \ldots, w_{i+v}$, have the prefix $w_{i_1} \ldots w_{i_n}1$, where $w_{i_1}, \ldots, w_{i_n} \in \mathcal{A}$. This ensures that that neither w_i nor $w_{i+1}, \ldots, w_{i+v}$ are prefixes of each other. The inductive step for this case follows.

The partitioning stops for $\mathcal{S}_0(j)$, since $|\mathcal{S}_0(j)| = 1$. For $\mathcal{S}_1(j)$, the partitioning is recursively invoked, leading again to either Case (*i*), (*ii*), or (*iii*).

(*iii*) As a result of the partitioning of $\mathcal{S}(j)$, $\mathcal{S}_0(j) = \{s_i, \ldots, s_{i+v-1}\}$ and $\mathcal{S}_1(j) = \{s_{i+v}\}$ are obtained, where $v = |\mathcal{S}(j)| - 1$. The analysis of this case is identical (except that the suffix of '0' becomes the suffix of '1' and vice versa) to the analysis of Case (*ii*) above, and omitted to avoid repetition.

From the three cases analyzed above, the induction is achieved. Thus, the procedure Fano(...) of Algorithm **Enhanced_Fano_Coding** generates a prefix code for every partitioning step 'j', and the result follows. □

Theorem 3.3. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq \ldots \geq p_m$, and the binary code alphabet. Suppose that $\mathcal{X}$ is encoded into $\mathcal{Y}$ using Algorithm **Enhanced_Fano_Coding**, then

(*i*) $\mathcal{X}$ is efficiently encoded into $\mathcal{Y}$, yielding an efficiency at least as that obtained using the conventional Fano's method.

(*ii*) If $\mathcal{Y}$ is decoded into $\mathcal{X}^*$ using Algorithm **Enhanced_Fano_Decoding**, then $\mathcal{X}^* = \mathcal{X}$.

Proof. We proceed by dividing the encoding and decoding algorithms into two sequential phases. The first consists of the traditional Fano coding itself, and the second is the code word rearrangement as per Rule 3.1.

Let us consider the encoding algorithm. The first phase involves the creation of a code, $\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$, using the traditional Fano coding.

In the second phase of the encoding, the code words of ϕ are rearranged using Rule 3.1, obtaining $\phi' : s_1 \rightarrow w'_1, \ldots, s_m \rightarrow w'_m$, where $w'_1, \ldots, w'_m$ is a permutation or rearrangement of $w_1, \ldots, w_m$, such that $\ell'_1 \leq \ldots \leq \ell'_m$.

We now observe that $w'_1, \ldots, w'_m$ is a sequence of code words that satisfies $\ell'_1 \leq \ldots \leq \ell'_m$. Since $\ell_1, \ldots, \ell_m$ is a rearrangement of the lengths $\ell'_1, \ldots, \ell'_m$, using the result of Theorem 3.1, it is clear that $\bar{\ell}' = \sum_{i=1}^{m} p_i \ell'_i \leq \sum_{i=1}^{m} p_i \ell_i = \bar{\ell}$. This proves the first assertion that the encoding scheme obtained after the rearrangement, ϕ', is at least as efficient as the original encoding scheme, ϕ.

We now consider Assertion (*ii*).

Suppose now that $\mathcal{Y}$ is decoded into $\mathcal{X}^*$ using Algorithm **Enhanced_Fano_Decoding**. In the first phase of the decoding process, a Fano tree is constructed using the conventional Fano coding, by procedure Fano(...), after which the actual decoding is achieved.

We, first of all, emphasize the duality of the encoding/decoding processes when a rearrangement is done. The effect of a rearrangement in the decoding process can be easily seen to obey the rule : Swapping two symbols in the tree, from $w_i \to s_i$ and $w_j \to s_j$, is equivalent to swapping the two symbols in the original scheme as $w_i \to s_j$ and $w_j \to s_i$. This is equivalent to swapping two code words in the *encoding* scheme generated in the encoding process : The swap of the pair $s_i \to w_i$ and $s_j \to w_j$ is equivalent to $s_i \to w_j$ and $s_j \to w_i$.

In the encoding, ϕ is generated by assigning (adding as a suffix) '0' to the code words corresponding to the symbols of the first sublist, $\mathcal{S}_0$, and '1' to the code words corresponding to the symbols of the second sublist, $\mathcal{S}_1$. This is equivalent to labeling the tree constructed in the decoder by assigning '0' to the left branches, and '1' to the right branches[4]. Using this tree and this predetermined labeling strategy to decode $\mathcal{Y}$, is equivalent to having a decoding scheme $\phi^{-1} : w_1 \to s_1, \ldots, w_m \to s_m$.

From Theorem 3.2, we know that ϕ is a prefix code. This uniquely decodability property, in turn, implies that for $w_1, \ldots, w_m$, no code word w_i is prefix of any other code word w_j, for all i and j, $i \neq j$. Clearly, any rearrangement of $w_1, \ldots, w_m$ also satisfies the prefix property, and so this is also true for the enhanced coding : $\phi : s_1 \to w'_1, \ldots, s_m \to w'_m$. Consequently, by proceeding from left to right, $\mathcal{X}$ is encoded into $\mathcal{Y}$ by using a uniquely decodable code, ϕ' [58]. This unique decodability property, in turn, implies that if $\phi'(\mathcal{X}) = \mathcal{Y}$, then $(\phi')^{-1}(\mathcal{Y}) = \mathcal{X}^\star = \mathcal{X}$, where $(\phi')^{-1} : w'_1 \to s_1, \ldots, w'_m \to s_m$. This proves the second assertion.

The theorem is thus proved. □

Remark 3.1. It is important to observe that, as pointed out in the proof of Theorem 3.3, when swapping two source alphabet symbols instead of the two corresponding code words, the code generated in the first phase preserves the property of being a canonical code, even after the rearrangement. This feature of Fano$^+$ is of fundamental importance, since, as mentioned earlier, canonical codes are desired because of their superior decoding speed, and the substantial memory savings that they yield. Moreover, this fundamental property of Fano$^+$ is performed without adding any extra computational

[4]This is an arbitrary labeling strategy. Any of the 2^{m-1} different strategies can also be used, whenever the encoder and the decoder are agreed on using the same labeling strategy.

File	Brief description
bib	Bibliographic files (refer format)
book1	Hardy: Far From The Madding Crowd
book2	Witten: Principles of Computer Speech
geo	Geophysical data
news	News batch file
obj1	Compiled code for Vax: compilation of progp
obj2	compiled code for Apple Macintosh: Knowledge support system
paper1	Witten, Neal and Cleary: Arithmetic coding for data compression
progc	C source code: compress version 4.0
progl	Lisp source code: system software
progp	Pascal source code: prediction by partial matching evaluation program
trans	Transcript of a session on a terminal

Table 3.1: A brief description of the files of the Calgary Corpus used to test the efficiency of the compression algorithms discussed in this research work.

complexity in either time or space.

We now present the empirical results demonstrating the efficiency of Fano$^+$, and comparing it to the relative efficiency of the Fano and Huffman algorithms.

3.3 Empirical Results

One of the set of files used to evaluate the performance of the compression algorithms presented in this research work was obtained from the University of Calgary. It has been universally used for many years as a standard test suite, and is known as the Calgary Corpus[5]. The other set of files was obtained from the University of Canterbury. This benchmark, known as the Canterbury Corpus, was proposed in 1997 as a replacement for the Calgary corpus test suite [10]. These files are widely used to test and compare the efficiency of different compression methods. They represent a wide range of different applications such as executable programs, spreadsheets, web pages, pictures, source code, postscript source, etc. A brief description of the files of the Calgary corpus and the Canterbury Corpus is given in Tables 3.1 and 3.2 respectively.

[5]Electronically available at `ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/`.

File	Brief description
alice29.txt	text (English text)
asyoulik.txt	Shakespeare (text format)
cp.html	Hyper Text Markup Language source code (Web page)
fields.c	Source code in C language
grammar.lsp	LISP source code
kennedy.xls	Microsoft Excel spreadsheet
lcet10.txt	Technical writing (text format)
plrabn12.txt	Poetry (text format)
ptt5	Picture number 5 from the CCITT Facsimile test files
xargs.1	GNU manual page for Unix systems

Table 3.2: Description of the files of the Canterbury corpus used as a benchmark for the compression algorithms presented in this research work.

In order to analyze the efficiency of the improved version of static Fano's method (Fano$^+$), we have conducted some experiments on files of the Calgary corpus and the Canterbury corpus. The empirical results obtained are displayed in Tables 3.3 and 3.4 respectively. The columns labeled 'Huffman' and 'Fano' correspond to the static Huffman's algorithm and the traditional static Fano's method respectively. The columns labeled 'Fano$^+$' correspond to the sibling-enhanced version of the static Fano's method introduced in this chapter. The columns labeled 'ℓ_y' represent the size (in bytes) of the compressed file. The columns labeled 'ρ' include the percentage of compression obtained by the different methods, as defined in (2.18). The last row contains the total for each column except for the column labeled 'ρ', for which the value of the cell corresponds to the *weighted average* of compression ratio.

Observe that the gain in percentage of compression is 0.06% and 0.07% on the files of the Calgary corpus and the Canterbury corpus respectively. Although the improved version of Fano's method does not guarantee the optimal encoding scheme, the weighted averages obtained from Fano$^+$ are significantly closer to those obtained by Huffman's algorithm. Observe also that in all the files, there is some gain in the compression ratio. This implies that in all the files encoded, $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$ was not satisfied by the encoding schemes generated by the traditional static Fano's method, and validates the results for such an enhancement.

File Name	Orig. Size (bytes)	Huffman		Fano		Fano$^+$	
		$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)
bib	111,261	73,003	34.38	73,133	34.26	73,010	34.37
book1	768,771	438,619	42.94	439,322	42.85	438,803	42.92
book2	610,856	368,587	39.66	369,691	39.47	369,537	39.50
geo	102,400	73,323	28.39	73,714	28.01	73,602	28.12
news	377,109	246,687	34.58	246,890	34.53	246,773	34.56
obj1	21,504	16,819	21.78	16,845	21.66	16,822	21.77
obj2	246,814	194,863	21.04	195,324	20.86	194,933	21.02
paper1	53,161	33,621	36.75	33,655	36.69	33,653	36.69
progc	39,611	26,189	33.88	26,355	33.46	26,326	33.53
progl	71,646	43,242	39.64	43,588	39.16	43,529	39.24
progp	49,379	30,480	38.27	30,527	38.17	30,501	38.23
trans	93,695	65,514	30.08	65,723	29.85	65,515	30.08
Total	2,547,207	1,610,947	36.76	1,614,767	36.61	1,613,004	36.68

Table 3.3: Empirical results obtained after compressing test files from the Calgary corpus by using the static Huffman's algorithm, the static Fano's method, and the sibling-enhanced version of the static Fano's method (Fano$^+$).

File Name	Orig. Size (bytes)	Huffman		Fano		Fano$^+$	
		$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)
alice29.txt	148,481	84,765	42.91	85,254	42.58	84,999	42.75
asyoulik.txt	125,179	76,010	39.27	76,195	39.13	76,124	39.18
cp.html	24,603	16,456	33.11	16,477	33.02	16,458	33.10
fields.c	11,150	7,295	34.56	7,354	34.03	7,349	34.08
grammar.lsp	3,721	2,397	35.56	2,402	35.44	2,397	35.56
kennedy.xls	1,029,744	463,300	55.00	465,368	54.80	464,612	54.88
lcet10.txt	419,235	244,124	41.76	244,197	41.75	244,141	41.76
plrabn12.txt	471,162	266,423	43.45	266,984	43.33	266,592	43.41
ptt5	513,216	107,027	79.14	107,138	79.12	107,075	79.13
sum	38,240	26,409	30.93	26,501	30.69	26,463	30.79
xargs.1	4,227	2,823	33.20	2,825	33.16	2,824	33.18
Total	2,788,958	1,297,029	53.49	1,300,695	53.36	1,299,034	53.42

Table 3.4: Details of the experiments performed with the static version of Huffman's algorithm, the traditional static Fano's method, and the sibling-enhanced version of static Fano's method on files of the Canterbury corpus.

3.4 Conclusions

In this chapter, we first showed that for the encoding schemes whose code words are not arranged in an increasing order of lengths, the corresponding coding tree does not satisfy the sibling property. To rectify this, we introduced an enhanced version of the static Fano's method, Fano$^+$.

Since Huffman's algorithm generates such an encoding scheme, we introduce the the concept of "code word rearrangement", derived from a result related to the ordering of two lists. By incorporating this result, we have devised an enhanced method for the static Fano coding.

The encoding algorithm, which includes a recursive procedure that generates the encoding scheme, has been formally presented. A similar procedure is incorporated in the decoding algorithm, which generates a Fano tree. We also derived an important property of the encoding schemes generated by Fano$^+$. This property is a direct consequence of the list rearrangement strategy that we incorporated. Finally, we formally showed that Fano$^+$ achieves lossless compression.

One of the most important features of Fano$^+$ is that it generates a canonical code without increasing the computational complexity in either time or space. As pointed out earlier, generating a canonical code using the Huffman coding requires fairly complicated processes. This is rendered straightforward in Fano$^+$. Note that this is significant since canonical codes are of fundamental importance in decoding, since they allow extremely fast decoding, and require half as much space as that used by the decoding tree.

Our empirical results on files of the Calgary corpus and the Canterbury corpus show that Fano$^+$ achieves percentages of compression which are almost optimal – very marginally below those of Huffman's algorithm. Furthermore, it is computationally superior, and is thus a desirable option when compared to the latter.

The extension of Fano$^+$ for multi-symbol code alphabets follows directly from the binary solution proposed. The main problem in dealing with multi-symbol code alphabets is that a more elaborate partitioning procedure is required, which we shall propose in Chapter 4.

Chapter 4

Greedy Adaptive Fano Coding

One of the major drawbacks of the adaptive Huffman coding scheme is that it requires the maintenance of an on-line Huffman tree which has to be updated each time a symbol is encoded. These memory requirements do not seem to be a serious problem for state-of-art computers, where a few kilobytes of information is insignificant. The real problem arises when we are dealing with Huffman coding involving higher-order modeling. Recall that for a kth-order model, m^k Huffman trees have to be maintained.

Fano's method, instead, proceeds by generating each code word directly from the list of probabilities. In the kth-order model, in which the required number of probability lists is m^k, the memory saving seems to be, obviously, considerable. Alternatively, only the probabilities for the contexts that have already appeared need to be maintained at any given time.

Although Fano's method, typically, generates a sub-optimal encoding scheme[1], the loss in compression ratio with respect to Huffman's algorithm is relatively small. However, as pointed out in Chapter 3, if the method is modified to also consider list rearrangement strategies, nearly optimal compression can be achieved.

In this chapter, we consider how Fano's method can be designed and implemented in an adaptive but greedy[2] manner.

[1]This is due to the fact that the optimal partitioning in Fano's method is NP-hard [152].

[2]This scheme, in itself, is greedy but not naive. It is done in a manner analogous to the way the adaptive Huffman method is designed. However, in the next chapter, we shall design a far more elegant algorithm which utilizes a new data structure – the adaptive Fano binary search tree.

Adaptive coding is important since the data is compressed by performing a single pass, as opposed to the static algorithms which require two passes – the first to learn the probabilities, and the second to accomplish the encoding. In adaptive coding, these two phases are carried out simultaneously in a single step. Conversely, adaptive coding is the best choice in many applications that require on-line compression such as in communication networks, LANs, internet applications, e-mail, ftp, e-commerce, etc.

In this chapter, we present a *greedy adaptive version* of Fano's method, specially suitable for applications in which memory constraints are tight. This chapter details the encoding and decoding algorithms, and the partitioning procedures suitable for the greedy adaptive method. It includes a rigorous analysis of the properties of the algorithm. It will be clear that the algorithm, though greedy, is still optimal.

We also present some empirical results obtained from running the greedy adaptive version of Fano's method and adaptive Huffman coding on files of the Calgary corpus and the Canterbury corpus.

We conclude this chapter with the extension of the greedy adaptive Fano coding to multi-symbol code alphabets. We introduce the corresponding greedy partitioning procedure that deals with consecutive partitionings possessing, what we have called, the *nearly-equal-probability property*. In this case, unlike the binary case, since the number of partitions is combinatorially large, the greedy algorithm is linear, but sub-optimal.

4.1 The Encoding Algorithm

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, where $m \geq 2$, with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, the code alphabet $\mathcal{A} = \{0, 1\}$, and the input sequence $\mathcal{X} = x[1] \ldots x[M]$. As in the adaptive Huffman coding (see Section 2.5.1 of Chapter 2), we assume that we start with arbitrary probabilities for the source symbols. The set $\mathcal{S}$ is considered as a list sorted in a non-increasing order of the occurrence probabilities. In order to hasten the encoding process, the probabilities of occurrence are considered as *frequency counters*[3], and initialized to unity.

The implementation of the encoding procedure for the greedy adaptive version

[3]We alternatively use the term "probabilities of occurrence" to also refer to these frequencies.

for Fano coding is shown in Algorithm **Greedy_Adaptive_Fano_Encoding**. At each time step, a list of symbols, $\mathcal{S}(k) = \{s_1(k), \ldots, s_m(k)\}$, is maintained. The starting list, $\mathcal{S}(1)$, is $\mathcal{S}$ itself. Whenever, $x[k]$, is encoded, $\mathcal{S}(k)$ is divided up into two sublists, and only the sublist in which $x[k]$ is contained, is maintained. When $r > 2$, the partitioning procedure is far from trivial. As discussed in Chapter 2, obtaining the optimal partitioning[4] is an NP-hard problem [152]. The case in which $r > 2$ is discussed later in this chapter.

As an alternative solution to that NP-hard problem, we assume that the components of each partitioning are consecutive in the original list. We formalize this property in the following definition.

Definition 4.1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the code alphabet $\mathcal{A} = \{0, 1\}$. Suppose that $\mathcal{S}$ is partitioned into two sublists $\mathcal{S}_0$ and $\mathcal{S}_1$. The partitioning is said to be *consecutive*, if and only if $\mathcal{S}_0 = \{s_1, \ldots, s_i\}$ and $\mathcal{S}_1 = \{s_{i+1}, \ldots, s_m\}$ for some $i = 2, \ldots, m-1$.

Observe that this definition does not permit empty sublists after the partitioning.

The frequency counter of $x[k]$ is updated by adding unity to its current value, and $\mathcal{S}(k)$ is kept sorted by interchanging $x[k]$ and the top-most symbol of $\mathcal{S}(k)$ whose frequency counter is less than that of $x[k]$. Note that two lists, $\mathcal{S}(k)$ and $\mathcal{P}(k)$, are kept in parallel. When interchanging two symbols in $\mathcal{S}(k)$, the corresponding components in $\mathcal{P}(k)$ must also be interchanged in order to maintain a correspondence between $\mathcal{S}(k)$ and $\mathcal{P}(k)$. Note also that the method of updating the statistical information applied here is trivial, i.e. adding unity[5] to the frequency counter of $x[k]$.

The partitioning procedure finds a sub-optimal partitioning in which the sublists obtained are as in Definition 4.1. The symbol $x[k]$ is the one being encoded at the kth step. The current range, $[t, b]$, contains the boundaries of $\mathcal{S}$ (or eventually a sublist of $\mathcal{S}$) in which $x[k]$ is contained. After the partitioning procedure is executed, these boundaries represent the sub-sublist in which $x[k]$ is contained. The variable p stands

[4]The optimal partitioning stands for a procedure that partitions $\mathcal{S}$ at every level of the Fano tree such that the encoding scheme generated with this tree has the minimum average code word length.

[5]Indeed, more advanced automaton-based probability updating techniques can also be used. The use of these techniques in data compression is currently being investigated.

Algorithm 9 Greedy_Adaptive_Fano_Encoding

Input: The source alphabet and probabilities, $\mathcal{S}$ and $\mathcal{P}$. The input sequence, $\mathcal{X}$.
Output: The output sequence, $\mathcal{Y}$.
Method:

```
procedure Partition(S, P, A : list; s : symbol; var t, b : integer; var p : real);
  i ← t; found ← false; p_temp ← 0
  while p_temp < p/2 do
    if s_i = s then found ← true endif
    p_temp ← p_temp + p_i; i ← i + 1
  endwhile
  if p/2 − p_temp + p_i/2 < 0 then
    i ← i − 1; p_temp ← p_temp − p_i
  endif
  if found then
    b ← i − 1; p ← p_temp; return 0
  else
    t ← i; p ← p − p_temp; return 1
  endif
endprocedure
S(1) ← S; P(1) ← P
for i ← 1 to m do    // Initialize the frequency counters
  p_i ← 1
endfor
j ← 1
for k ← 1 to M do    // For every symbol of the source sequence
  t ← 1; b ← m; p ← Σ_{i=1}^{m} p_i
  while b > t do
    y[j] ← Partition(S(k), P(k), A, x[k], t, b, p)
    j ← j + 1
  endwhile
  Increment the weight of x[k].
  Swap x[k] and the top-most symbol in S(k) whose weight is less than
      that of x[k]. (Do the same updating for P(k))
endfor
end Algorithm Greedy_Adaptive_Fano_Encoding
```

for the sum of probabilities of those symbols between t and b. After executing the partitioning procedure, this variable contains the sum of probabilities of the resulting sub-sublist. If the symbol being encoded is found in the first sublist, a '0' is sent to the output sequence as a suffix (i.e. exactly as described earlier in Chapter 3), otherwise a '1' is sent to the output.

4.2 The Decoding Algorithm

The decoding procedure is similar to the encoder. Instead of probabilities, we again keep a frequency counter for each symbol, which records the number times that this symbol has appeared so far in the input sequence. The decoding procedure is detailed in Algorithm **Greedy_Adaptive_Fano_Decoding**.

Note that, as discussed earlier, no extra information other than the encoded sequence is provided to the encoder. This is, in fact, an important advantage over the static methods which require that some extra information (such as the symbol probabilities, the decoding tree, etc.) be stored in the overhead of the encoded sequence.

Each time $y[k]$ is read from the input, $\mathcal{S}(j)$ and $\mathcal{P}(j)$ are partitioned into two as nearly-equal as possible (in probability) sublists[6]. The sublist that corresponds to $y[k]$ is chosen based on a labeling scheme which turns out to be exactly the same as the one used by the encoder. A partitioning procedure which is similar to that invoked by the encoder is now utilized to render the two processes synchronized.

The next symbol is read from the encoded sequence, and the partitioning continues until a sublist with a single symbol is obtained. This symbol is extracted as part of the reconstruction of the original input sequence. The consistency between the encoder and the decoder is achieved because they both encode/decode the kth symbol using the statistical information obtained from the $(k-1)^{th}$ step, which is known by both the encoder and the decoder.

We now observe some important properties which are crucial when implementing

[6]Observe that since the encoder and the decoder arbitrarily start with the same lists, $\mathcal{S}(1)$ and $\mathcal{P}(1)$, by using the same updating procedure, the lists that they maintain, $\mathcal{S}(j)$ and $\mathcal{P}(j)$, are always identical (synchronized) at the sender/receiver ends.

Algorithm 10 Greedy_Adaptive_Fano_Decoding

Input: The encoded sequence, $\mathcal{Y}$. The source alphabet and probabilities, $\mathcal{S}$ and $\mathcal{P}$.
Output: The source sequence, $\mathcal{X}$.
Method:

```
procedure Partition(S, P, A : list; a : symbol; var t, b : integer; var p : real);
  p_temp ← 0; i ← t
  while p_temp < p/2 do
    p_temp ← p_temp + p_i; i ← i + 1
  endwhile
  if p/2 − p_temp + p_i/2 < 0 then
    i ← i − 1; p_temp ← p_temp − p_i
  endif
  if a = 0 then
    b ← i − 1; p ← p_temp
  else
    t ← i; p ← p − p_temp
  endif
endprocedure
S(1) ← S; P(1) ← P
for i ← 1 to m do
  p_i ← 1
endfor
k ← 1; t ← 1; b ← m; p ← Σ_{i=1}^{m} p_i
for j ← 1 to R do
  Partition(S(k), P(k), A, y[j], t, b, p)
  if t = b then
    x[k] ← s_t(k)
    Increment the weight of x[k]
    Swap x[k] and the top-most symbol in S(k) whose counter is less than
        that of x[k]. (Do the same updating for P(k))
    k ← k + 1; t ← 1; b ← m; p ← Σ_{i=1}^{m} p_i
  endif
endfor
end Algorithm Greedy_Adaptive_Fano_Decoding
```

the encoding and decoding algorithms. First, the encoder and the decoder must use the same labeling scheme. Second, they both have to start with the same initial probabilities. Third, they must use the same probability updating procedure. Finally, the encoder uses $\mathcal{P}(k)$ to encode $x[k]$, which, obviously, is known by the decoder too.

4.3 Correctness of the Greedy Adaptive Fano Coding

In this section, we analyze the greedy adaptive Fano coding algorithms with respect to its lossless data compression properties. We shall show that given any source sequence, the output sequence generated by Algorithm **Greedy_Adaptive_Fano_Encoding** has enough information to exactly reproduce the original input sequence using Algorithm **Greedy_Adaptive_Fano_Decoding**. We assume that both algorithms partition the list by using identical partitioning rules, and update the probabilities of occurrence by using the same updating procedure. We also show that on-line compression is achieved, which, as mentioned earlier, is quite useful in communications and networking applications, such as those involving the internet, ftp, e-commerce, e-mail, audio and video streaming, etc.

Theorem 4.1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the code alphabet $\mathcal{A} = \{0, 1\}$. Suppose that the input sequence $\mathcal{X} = x[1] \ldots x[M]$ is encoded by using Algorithm **Greedy_Adaptive_Fano_Encoding**, yielding an output sequence $\mathcal{Y} = y[1] \ldots y[R]$. If $\mathcal{Y}$ is decoded by using Algorithm **Greedy_Adaptive_Fano_Decoding** into $\mathcal{X}^\star = x^\star[1] \ldots x^\star[T]$, then $\mathcal{X}^\star = \mathcal{X}$ (i.e. $T = M$, and $x^\star[i] = x[i]$, $\forall i$, $i = 1, \ldots, M$).

Proof. We prove the result by invoking a double induction. We perform an *outer* induction on the number of encoding steps, assuming that $x[k]$ is encoded at step k, and for each encoding step, we perform an *inner* induction on the number of partitioning steps 'j'.

At the kth encoding step, both the encoder and the decoder can be perceived to be composed of two phases:

(i) Task of Encoder: Encode $x[k]$ using $\mathcal{S}(k)$ and $\mathcal{P}(k)$.

Task of Decoder: Decode the relevant sequence into $x^*[k]$ using $\mathcal{S}(k)$ and $\mathcal{P}(k)$.

This part is proved by induction on the number of partitioning steps 'j'.

(*ii*) Task of both Encoder and Decoder: Update $\mathcal{S}(k)$ and $\mathcal{P}(k)$ to yield $\mathcal{S}(k+1)$ and $\mathcal{P}(k+1)$. The steps involved in both the encoder and decoder are identical.

The outer induction based on the number of encoding/decoding steps follows.

Basis step for Outer Induction: Both the encoder and the decoder start with the same lists of symbols and probabilities, $\mathcal{S}(1)$ and $\mathcal{P}(1)$, where $p_i = 1$, for $i = 1, \ldots, m$. We have to prove that $x^*[1] = x[1]$.

We achieve this part of the proof using an induction on the number of partitioning steps 'j'. We prove that, for every $j \geq 1$, the encoder and the decoder obtain the same values of p, t, and b.

Basis step for $x[1]$: We first prove that the encoder and the decoder obtain the same values of p, t, and b, for $j = 1$.

By initialization, both the encoder and the decoder start with the same values of p, t, and b. They also find the value that i takes after $p_{\text{temp}} \geq \frac{p}{2}$. They "backtrack" if $\frac{p}{2} - p_{\text{temp}} + \frac{p_i}{2} < 0$, implying that they subtract unity from i.

We now have two mutually exclusive and exhaustive cases for $x[1]$.

(a) $x[1]$ is in $\mathcal{S}_0(1)$. In the encoder, b is set to $i-1$, p is set to p_{temp}, t maintains its previous value, and '0' is sent to the output. The decoder receives a '0', at which time it sets b to $i-1$, sets p to p_{temp}, and does not modify the value of t.

(b) $x[1]$ is *not* in $\mathcal{S}_0(1)$. The encoder sets t to i, and p to $p - p_{\text{temp}}$. It does not modify the value of b, and sends a '1' to the output. A '1' is received by the decoder, t is set to i, p is set to $p - p_{\text{temp}}$, and b remains unchanged.

This proves the basis step because both achieve the identical values of p, t and b, implying that they both have identical sublists for the next step, and thus achieve encoding and decoding losslessly.

Inductive hypothesis for $x[1]$: We assume that the result holds for $j = n$. This implies that $b = i - 1$, and $p = p_{\text{temp}}$ if $x[1]$ is in $\mathcal{S}_0(1)$. Conversely, $p = p - p_{\text{temp}}$ and $t = i$ if $x[1]$ is in $\mathcal{S}_1(1)$.

The inductive hypothesis of the inner induction states that after the nth partitioning step, the encoder and the decoder obtain the same values of p, t, and b. The encoder and the decoder set i to t, and find the value of i after $p_{\text{temp}} \geq \frac{p}{2}$. Besides, both the encoder and the decoder subtract unity from i if $\frac{p}{2} - p_{\text{temp}} + \frac{p_i}{2} < 0$.

Inductive step for $x[1]$: We now have to prove that the result holds of $j = n + 1$.

We have two mutually exclusive and exhaustive cases for $x[1]$:

(a) If $x[1]$ is in $\mathcal{S}_0(1)$ in the encoder, b is set to $i - 1$, p is set to p_{temp}, and '0' is sent to the output. The decoder receives a '0', sets b to $i - 1$ and p to p_{temp}.

(b) When $x[1]$ is *not* in $\mathcal{S}_0(1)$ in the encoder, t is set to i, p is set to $p - p_{\text{temp}}$, and '1' is sent to the output. The decoder receives a '1', sets t to i, and p to $p - p_{\text{temp}}$.

Again, since both the encoder and the decoder have identical values of p, t and b. This implies that they both have identical sublists for the next step, and achieve encoding and decoding losslessly. The inner induction thus follows for $x[1]$. In terms of clarification, we see that the encoder and the decoder find the same values of p, t, and b for every partitioning step j, where $j \geq 1$. In the encoder, the partitioning stops when $b \leq t$ in the encoder. This will occur when there is only one element in $\mathcal{S}(1)$, $x[1]$, determined by $t = b$. In the decoder, when $t = b$, which from the inner induction occurs at the same 'j' for both the encoder and the decoder, $s_t(1)$ is recovered. Since t has

the same value for the encoder and the decoder, $s_t(1) = x^*[1] = x[1]$. This completes the basis step for the outer induction on the number of encoding steps 'k'.

On the other hand, both the encoder and the decoder increment the weight of $x[1]$ and update $\mathcal{S}(1)$ and $\mathcal{P}(1)$ by swapping $x[1]$ with the top-most symbol in $\mathcal{S}(1)$ whose weight is less than that of $x[1]$, resulting in the same lists, $\mathcal{S}(2)$ and $\mathcal{P}(2)$, for both the encoder and the decoder.

We now continue the outer level of the induction – namely the processing of $x[v]$.

Inductive hypothesis for Outer Induction: We now assume that lossless compression is achieved for a particular number of encoding steps, $k = v$, where $v > 1$. This means that the encoder encodes $x[v]$ using $\mathcal{S}(v)$ and $\mathcal{P}(v)$, and the decoder successfully obtains $x^*[v] = x[v]$. They also update $\mathcal{S}(v)$ and $\mathcal{P}(v)$ using identical rules, obtaining the same symbol and probability lists, $\mathcal{S}(v+1)$ and $\mathcal{P}(v+1)$.

Inductive step for Outer Induction: We now have to prove that lossless compression is achieved for $k = v + 1$.

From the inductive hypothesis of the outer induction, we see that:

(a) The encoder and the decoder obtain the same lists $\mathcal{S}(v+1)$ and $\mathcal{P}(v+1)$.

(b) $x^*[1] = x[1], \ldots, x^*[v] = x[v]$.

We have to prove that $x^*[v+1] = x[v+1]$. We prove *this* by an induction on the number of partitioning steps 'j' for this symbol.

<u>Basis step for $x[v+1]$</u>: In this case, we will see that the encoder and the decoder obtain the same values of p, t, and b, for $j = 1$.

From the inductive hypothesis of the outer induction we see that the encoder and the decoder start with the same values for p, t, and b. These are $\sum_{i=1}^{m} p_i$, 1, and m respectively.

Subsequently, the encoder and the decoder find the value that i has after $p_{\text{temp}} \geq \frac{p}{2}$, and they both "backtrack" one step if $\frac{p}{2} - p_{\text{temp}} + \frac{p_i}{2} < 0$ – they subtract unity from i.

The values of p, t and b are modified depending on the following two mutually exclusive and exhaustive cases in which $x[v+1]$ falls.

(a) When $x[v+1]$ *is* in $\mathcal{S}_0(v+1)$ in the encoder, b is set to $i-1$, p is set to p_{temp}, t maintains its previous value, and '0' is sent to the output. This '0' is received by the decoder, which sets b to $i-1$, p to p_{temp}, and does not modify the value of t.

(b) If $x[v+1]$ is *not* in $\mathcal{S}_0(v+1)$, the encoder sets t to i, sets p to $p - p_{\text{temp}}$, does not modify the value of b, and sends a '1' to the output. The decoder receives a '1', sets t to i, and p to $p - p_{\text{temp}}$. As in the encoder, b remains unchanged.

Since both the encoder and the decoder achieve identical values for p, t and b, they maintain identical sublists for the next steps, and thus achieve lossless compression and decompression. The basis step of the inner induction thus follows.

Inductive hypothesis for $x[v+1]$: We assume that the result holds for $j = n$, where j is the number of partitionings, and $n > 1$. This implies that p, b, and t have the same values in the encoder and the decoder :

(a) $p = p_{\text{temp}}$ and $b = i-1$, if $x[v+1]$ is in $\mathcal{S}_0(v+1)$, and

(b) $p = p - p_{\text{temp}}$ and $t = i$, if $x[v+1]$ is in $\mathcal{S}_1(v+1)$.

Furthermore, in the inductive hypothesis of the inner induction, we assume the encoder and the decoder set i to t, and find the value of i after $p_{\text{temp}} \geq \frac{p}{2}$. Besides both the encoder and the decoder subtract unity from i if $\frac{p}{2} - p_{\text{temp}} + \frac{p_i}{2} < 0$.

Inductive step: We now have to prove that the encoder and the decoder have the same values of p, t and b, for $j = n+1$.

At this point, we have two mutually exclusive and exhaustive cases for $x[v+1]$:

(a) When $x[v+1]$ is in $\mathcal{S}_0(v+1)$, the encoder sets b to $i-1$, sets p to p_{temp}, and sends a '0' to the output. The decoder receives a '0', and sets b to $i-1$, and p to p_{temp}.

(b) In this case, $x[v+1]$ is in $\mathcal{S}_1(v+1)$ in the encoder, and so, t is set to i, p is set to $p - p_{\text{temp}}$, and a '1' is sent to the output. The decoder receives a '1', and sets t to i, and p to $p - p_{\text{temp}}$.

This completes the inner induction on the number of partitioning steps 'j' for $x[v+1]$ because for every $j \geq 1$, the encoder and the decoder obtain the same values of p, t and b. They thus maintain identical sublists and consequently achieve lossless compression.

The partitioning is performed in the encoder until $b \leq t$. This will happen in the decoder when $t = b$, and $s_t(v+1)$ is recovered. Since t has the same value in both the encoder and the decoder, the symbol recovered is $s_t(v+1) = x^*[v+1] = x[v+1]$.

This completes the outer induction on the number of encoding steps, i.e. the number of symbols encoded and decoded. This result holds for all $v \geq 1$, and the theorem is proved.

□

Theorem 4.2. Under the conditions of Theorem 4.1, encoding using Algorithm **Greedy_Adaptive_Fano_Encoding**, and decoding using Algorithm **Greedy_Adaptive_Fano_Decoding** achieve *on-line compression.*

Proof. From the inductive proof given in Theorem 4.1 on the number of encoding steps, we see that for all k, $k = 1, \ldots, M$, $x[k]$ is encoded into a binary sequence $y[j] \ldots y[j+\ell_k]$, where ℓ_k is the length of the code word of $x[k]$. This binary sequence is simultaneously decoded into $x[k]$. Moreover, each bit, $y[j]$, sent to the output is immediately processed by the decoder in a prefix manner. Consequently, encoding by using Algorithm **Greedy_Adaptive_Fano_Encoding** and decoding by using Algorithm **Greedy_Adaptive_Fano_Decoding** achieve *on-line compression.* □

4.4 Complexity of the Greedy Fano Algorithms

In this section, we analyze the *worst-case* computational complexity of the greedy adaptive Fano coding algorithms.

We shall first analyze the time complexity for the encoding algorithm. The first **for** loop is executed m times. Inside the second **for** loop, which is executed M times, there is a **while** loop. This loop contains the invocation of the partitioning procedure which takes $|\mathcal{S}_0|$ steps. Hence, if $x[k]$ is in $\mathcal{S}_0$, the total number of steps to encode $x[k]$ is not greater than m, because in the worst case, it would take up to $|\mathcal{S}_0|$ more steps to find $x[k]$. If $x[k]$ is not in $\mathcal{S}_0$, it is searched in $\mathcal{S}_1$, which takes up to m steps (in the worst case, when $x[k]$ is the last element of $\mathcal{S}$). Therefore, given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ and the input sequence $\mathcal{X} = x[1] \ldots x[M]$, in the worst case, the number of steps that Algorithm **Greedy_Adaptive_Fano_Encoding** takes to generate the output is $O(mM)$ steps.

The time complexity of Algorithm **Greedy_Adaptive_Fano_Decoding** is derived as in the encoding algorithm. The first **for** loop, which initializes the probabilities, takes m steps. The main **for** loop, which takes R steps, invokes the partitioning procedure. All the partitioning steps for encoding $x[k]$ take up to m steps, and there are $\frac{R}{l} \approx M$ symbols in $\mathcal{Y}$. Therefore, given the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, Algorithm **Greedy_Adaptive_Fano_Decoding** decodes an encoded sequence $\mathcal{Y} = y[1] \ldots y[R]$ in $O(mM)$ steps.

The space complexity of Algorithms **Greedy_Adaptive_Fano_Encoding** and **Greedy_Adaptive_Fano_Decoding** is relatively small, since they require the maintenance of only a list of symbols and a list of probabilities. Each symbol and probability require constant space. Thus, the space complexity of these algorithms is $O(m)$.

Observe that the Huffman tree also requires $O(m)$ space. But the maintenance and overhead is significantly more than for the Fano "tree" – since the latter is maintained essentially as a list. Thus, while the Huffman tree requires $O(m)$ space, it still needs to maintain $2m - 1$ nodes, and each node has a symbol, a weight, a pointer to its parent, and pointers to its left and right children. This is significantly more than what is

required by the corresponding data structure used in the greedy adaptive Fano coding. For example, consider the case when the source alphabet is the ASCII alphabet. If one byte is required for the symbol, and two bytes for each of the other fields, it implies that the adaptive Huffman coding data structure requires 4,599 bytes, in contrast to the 768 bytes required by the greedy adaptive Fano coding – which is approximately a six-fold advantage. This advantage increases significantly if higher-order models and dictionary-based structures are used.

4.5 Empirical Results

In order to test the compression power and the compression/decompression speed, we have implemented Algorithm **Greedy_Adaptive_Fano_Encoding** and Algorithm **Greedy_Adaptive_Fano_Decoding**, using the state-of-art Java programming language [39, 67, 144]. We have also implemented the adaptive version of Huffman's algorithm as presented in [75]. To compare the greedy adaptive version of Fano's method with the adaptive Huffman coding, we ran both prototypes on files of the Calgary corpus and Canterbury corpus (detailed in Tables 3.1 and 3.2 respectively).

The empirical results obtaining from these runs are cataloged in Tables 4.1 and 4.2 respectively. The name and the original size of the files are given in the first and second columns respectively. The columns labeled 'ρ' correspond to the compression ratio, as defined in (2.18). The columns labeled '$\ell_{\mathcal{Y}}$' represent the size of the compressed file. The columns labeled 'Enc' and 'Dec' contain the time (in seconds) used to compress the original file and decompress the compressed file respectively.

Observe that the greedy adaptive Fano's method compresses with a marginally less efficiency than the adaptive Huffman coding. Huffman coding compressed just 0.14% more than Fano's method on the files of the Calgary corpus. This difference is slightly higher, 0.2%, on the files of the Canterbury corpus. This difference in favor of Huffman coding is present in all the files but the file called "xargs.1", in which Fano's method compressed two bytes more than Huffman coding. This behavior, which seems illogical since Huffman coding achieves optimal compression if the probabilities were exact, is because both methods use different probability updating procedures.

File Name	Orig. Size (bytes)	Huffman's algorithm ℓ_y (bytes)	ρ (%)	Time Enc. (sec.)	Time Dec. (sec.)	Fano's method ℓ_y (bytes)	ρ (%)	Time Enc. (sec.)	Time Dec. (sec.)
bib	111,261	73,037	34.35	2.51	0.41	73,191	34.21	1.31	0.86
book1	768,771	438,745	42.92	5.92	2.62	439,457	42.83	6.97	5.10
book2	610,856	368,652	39.64	5.62	2.07	369,701	39.47	6.04	4.41
geo	102,400	72,727	28.97	2.53	0.47	73,052	28.66	2.24	0.85
news	377,109	246,710	34.57	4.20	1.44	247,159	34.45	4.58	2.88
obj1	21,504	16,198	24.67	1.92	0.12	16,300	24.20	0.49	0.20
obj2	246,814	194,325	21.26	3.63	1.16	194,462	21.21	5.39	2.43
paper1	53,161	33,585	36.82	2.03	0.19	33,594	36.80	0.57	0.38
progc	39,611	26,142	34.00	1.93	0.16	26,285	33.64	0.50	0.28
progl	71,646	43,242	39.64	2.17	0.25	43,554	39.20	0.71	0.52
progp	49,379	30,471	38.29	2.04	0.18	30,713	37.80	0.54	0.35
trans	93,695	65,486	30.10	2.43	0.37	65,547	30.04	1.41	0.77
Total	2,547,207	1,609,320	36.82	36.93	9.44	1,613,015	36.68	30.75	19.03

Table 4.1: An empirical comparison of the greedy adaptive version of Fano's method and the adaptive Huffman coding which were tested on files of the Calgary corpus.

File Name	Orig. Size (bytes)	Huffman's algorithm ℓ_y (bytes)	ρ (%)	Time Enc. (sec.)	Time Dec. (sec.)	Fano's method ℓ_y (bytes)	ρ (%)	Time Enc. (sec.)	Time Dec. (sec.)
alice29.txt	148,481	84,858	42.84	2.49	0.48	85,239	42.59	1.20	0.99
asyoulik.txt	125,179	76,095	39.21	2.37	0.42	76,287	39.05	1.25	0.89
cp.html	24,603	16,416	33.27	1.87	0.10	16,438	33.19	0.30	0.19
fields.c	11,150	7,213	35.31	1.77	0.05	7,256	34.92	0.14	0.08
grammar.lsp	3,721	2,318	37.70	1.70	0.02	2,334	37.29	0.05	0.02
kennedy.xls	1,029,744	462,884	55.04	6.71	2.90	466,801	54.66	11.72	5.76
lcet10.txt	419,235	244,210	41.74	4.18	1.41	244,322	41.72	3.83	2.84
plrabn12.txt	471,162	266,528	43.43	4.40	1.52	267,051	43.32	4.20	3.08
ptt5	513,216	106,812	79.18	2.57	0.67	106,929	79.16	1.68	1.29
sum	38,240	25,820	32.48	1.93	0.18	26,192	31.50	0.66	0.30
xargs.1	4,227	2,758	34.77	1.74	0.02	2,756	34.81	0.06	0.03
Total	2,788,958	1,295,912	53.53	31.73	7.77	1,301,605	53.33	25.09	15.47

Table 4.2: Empirical Results obtained after running the adaptive Huffman coding and greedy adaptive Fano's method on files of the Canterbury corpus.

Regarding the compression time, we observe that in some cases, Fano's method is faster than Huffman coding. This is reasonable since their computational complexity is the same. In the case of the decompression, Huffman coding is faster than Fano's method. This is due to the fact that Huffman coding decodes in a top down fashion, avoiding the double pass required in the encoding process.

Finally, note that although both techniques require the same time complexity, Fano's method uses only the list of symbols and their frequency counters. Huffman's algorithm, instead, maintains a list of symbols and a Huffman tree, for which the space required is about six times as much for the case of the ASCII alphabet.

4.6 Greedy Adaptive Fano Coding for Multi-symbol Code Alphabets

The extension of the adaptive version of Fano's method for a code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$, where $r > 2$, is not straightforward. First of all, we observe that, Huffman's algorithm requires that the number of source alphabet symbols is $m = r + \kappa(r-1)$ for some positive integer κ. Fano's method also requires that this constraint be satisfied. However, this showed be satisfied not only for the source alphabet, but also for all the sublists obtained after partitioning.

To clarify issues, we consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, the probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. We now design Algorithm **Greedy_Adaptive_Fano_Encoding** which requires that $\mathcal{S}$ and $\mathcal{P}$ be partitioned into r sublists $\{\mathcal{S}_1, \ldots, \mathcal{S}_r\}$, such that $|\mathcal{S}_j| = \kappa(r-1)+1$ for some positive integer κ. The output symbol, $y[j]$, is to be a_i where $x[k] \in \mathcal{S}_i$. This partitioning scheme is also required by Algorithm **Greedy_Adaptive_Fano_Decoding**. The partitioning procedure that takes care of these constraints is designed below.

4.6.1 Nearly-Equal-Probability Partitioning

In order to design a suitable partitioning procedure, just as in the binary-alphabet case, we assume that the components of each partitioning are consecutive in the original

partitioned list. This is formally stated in the following definition.

Definition 4.2. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, where $m = r + \kappa(r-1)$ for some integer $\kappa \geq 0$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Suppose that $\mathcal{S}$ is partitioned into $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r\}$. The partitioning is said to be *consecutive*, if and only if $\mathcal{S}_1 = \{s_1, \ldots, s_{i_1}\}, \mathcal{S}_2 = \{s_{i_1+1}, \ldots, s_{i_2}\}, \ldots, \mathcal{S}_r = \{s_{i_{r-1}+1}, \ldots, s_m\}$, where $1 \leq i_1 < i_2 < \ldots < i_{r-1} < m$, and $|\mathcal{S}_j| = \kappa_j(r-1)+1$ for integers $\kappa_j \geq 0$, $j = 1, \ldots, r$.

When considering consecutive partitionings, the number of possible such partitionings is given by $\frac{(\frac{m-r}{r-1}+r-1)!}{(\frac{m-r}{r-1})!(r-1)!}$, where m is the size of the source alphabet, and r is the size of the code alphabet. This is stated and proved in the following theorem.

Theorem 4.3. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet and $\mathcal{A} = \{a_1, \ldots, a_r\}$ be the code alphabet, where $m = r + \kappa(r-1)$ for some integer $\kappa \geq 0$, and $r \geq 2$. The number of possible consecutive partitionings is given by:

$$\binom{\frac{m-r}{r-1} + r - 1}{r-1} = \frac{(\frac{m-r}{r-1} + r - 1)!}{(\frac{m-r}{r-1})!(r-1)!}. \tag{4.1}$$

Proof. Using the definition of consecutive partitionings (Definition 4.2), $\mathcal{S}$ is partitioned into:

$\mathcal{S}_1 = \{s_1, \ldots, s_{i_1}\}, \mathcal{S}_2 = \{s_{i_1+1}, \ldots, s_{i_2}\}, \ldots, \mathcal{S}_r = \{s_{i_{r-1}+1}, \ldots, s_m\}$,
where $1 \leq i_1 < i_2 < \ldots < i_{r-1} < i_m$, and $|\mathcal{S}_j| = \kappa_j(r-1)+1$, for integers $\kappa_j \geq 0$, $j = 1, \ldots, r$.

We now observe the following:

(*i*) Since $\mathcal{S}_1$ has $\kappa_1(r-1)+1$ symbols for some integer $\kappa_1 \geq 0$, and s_1 and s_{i_1} are the first and the last elements of $\mathcal{S}_1$ respectively, it implies that $i_1 = \kappa_1(r-1)+1$, where $\kappa_1 = 0, \ldots, \kappa$.

(*ii*) Again, $\mathcal{S}_2$ has also $\kappa_2(r-1)+1$ symbols, for some integer $\kappa_2 \geq 0$. Thus, s_{i_1+1} and s_{i_2} are the first and the last elements of $\mathcal{S}_2$. This implies that $i_2 = \kappa_2(r-1)+2$, where $\kappa_2 = 0, \ldots, \kappa$, and $\kappa_2 \geq \kappa_1$.

Continuing in the same fashion, we have:

(r-1) Since $\mathcal{S}_{r-1}$ has also $\kappa_{r-1}(r-1)+1$ symbols, for some integer $\kappa_{r-1} \geq 0$, and $s_{i_{r-2}+1}$ and $s_{i_{r-1}}$ are the first and last elements of $\mathcal{S}_{r-1}$ respectively, we have $i_{r-1} = \kappa_{r-1}(r-1)+(r-1)$, where $\kappa_{r-1} = 0, \ldots, \kappa$, and $\kappa_{r-1} \geq \kappa_{r-2}$.

Let i_j be a *valid partitioning index*. Also, let the jth sublist resulting from the partitioning be $\mathcal{S}_j = \{s_{i_{j-1}+1}, \ldots, s_{i_j}\}$.

From the arguments above, we observe that:

$$\kappa_j = \kappa_j(r-1)+j, \text{ where } \kappa_j = 0, \ldots, \kappa, \text{ and } j = 1, \ldots, r-1.$$

We also know that $1 \leq i_1 < i_2 < \ldots < i_{r-1} < m$, which implies that:

$$1 \leq \kappa_1(r-1)+1 < \kappa_2(r-1)+2 < \ldots < \kappa_{r-1}(r-1)+(r-1) < m\,. \tag{4.2}$$

Subtracting unity from all the terms of the inequality, we have:

$$0 \leq \kappa_1(r-1) < \kappa_2(r-1)+1 < \ldots < \kappa_{r-1}(r-1)+(r-2) < m-1\,. \tag{4.3}$$

Since all the quantities involved in the inequalities are integers, (4.3) can also be written as follows:

$$0 \leq \kappa_1(r-1) \leq \kappa_2(r-1) \leq \ldots \leq \kappa_{r-1}(r-1) \leq m-r\,. \tag{4.4}$$

Dividing all the terms of (4.4) by $r-1$, and using the fact that $m = r+\kappa(r-1)$ (which implies that $\kappa = \frac{m-r}{r-1}$), we have:

$$0 \leq \kappa_1 \leq \kappa_2 \leq \ldots \leq \kappa_{r-1} \leq \kappa\,. \tag{4.5}$$

Therefore, the number of possible consecutive partitionings is given by the number of ways of choosing a set of $r-1$ integers, $\{\kappa_1, \kappa_2, \ldots, \kappa_{r-1}\}$, from a set of $\kappa+1$

values, $\{0, 1, \ldots, \kappa\}$, where repetitions are allowed. This number is given by ([129], pp. 286–291):

$$\binom{\kappa + r - 1}{r - 1}. \tag{4.6}$$

Since $\kappa = \frac{m-r}{r-1}$, equation (4.6) can be written as follows:

$$\binom{\frac{m-r}{r-1} + r - 1}{r - 1} = \frac{\left(\frac{m-r}{r-1} + r - 1\right)!}{\left(\frac{m-r}{r-1}\right)!(r-1)!}.$$

Hence the theorem. □

After determining the number of all possible consecutive partitionings, we are only interested in the ones that satisfy the principles of Fano coding, i.e. we would like to ensure that the sums of the probabilities of the sublists resulting from the partitioning are as nearly equal as possible. As in the binary-alphabet case, we define this property for a multi-symbol code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$, where $r \geq 2$. This property is formally stated in the following definition.

Definition 4.3. Let $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r\}$ be a set obtained by a consecutive partitioning on $\mathcal{S}$. Let q_j be the sum of probabilities of $\mathcal{S}_j$, and q be the sum of probabilities of $\mathcal{S}$. $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r\}$ has the *nearly-equal-probability property*, if and only if

$$d(\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_r\}) = \sum_{k=1}^{r} \left| q_j - \frac{q}{r} \right|^{\alpha} \tag{4.7}$$

is minimal, where α is a real number such that $\alpha \geq 1$, and $d(.)$ denotes the *distance function.*

We now introduce an example that clarify issues.

Example 4.1. Consider the source alphabet $\mathcal{S} = \{a, b, c, d, e, f, g\}$ whose probabilities of occurrence are $\mathcal{P} = [.28, .22, .2, .1, .1, .05, .05]$, and the code alphabet $\mathcal{A} = \{0, 1, 2\}$.

Using (4.1), we see that there are $\binom{\frac{7-3}{3-1}+3-1}{3-1} = 6$ possible consecutive partitionings. The six partitionings, the sums of their corresponding probabilities, and the distance as calculated from (4.7) for $\alpha = 1$, are listed in the following table.

$\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$	$q_j = \sum_{s_i \in \mathcal{S}_j} p_i, j = 1, 2, 3$	$d(\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\})$
$\{\{a\}, \{b\}, \{c, d, e, f, g\}\}$	**.28, .22, .5**	**0.33**
$\{\{a\}, \{b, c, d\}, \{e, f, g\}\}$	.28, .52, .2	0.37
$\{\{a\}, \{b, c, d, e, f\}, \{g\}\}$	.28, .67, .05	0.67
$\{\{a, b, c\}, \{d\}, \{e, f, g\}\}$	.7, .1, .2	0.73
$\{\{a, b, c\}, \{d, e, f\}, \{g\}\}$	.7, .25, .05	0.73
$\{\{a, b, c, d, e\}, \{f\}, \{g\}\}$	.9, .05, .05	1.13

Observe that there is one optimal consecutive partitioning (i.e. the one that satisfies the nearly-equal-probability property stated in Definition 4.3), and is the one found in the first row of the table, namely, $\{\{a\}, \{b\}, \{c, d, e, f, g\}\}$, whose distance is $\sum_{j=1}^{3} \left|q_j - \frac{1}{3}\right| = 0.33$. The optimal algorithm would attempt to compute this partitioning. This can be achieved by a brute-force polynomial-time algorithm presented momentarily. We shall later show that by a linear searching scheme, a sub-optimal, but accurate, partitioning can be obtained. □

We now introduce a "brute force" algorithm that searches the space of *all* possible consecutive partitionings, and computes those that satisfy the nearly-equal-probability property, i.e. the ones whose distance is minimal. The formal procedure is shown below in Algorithm **All_Consecutive_Partitionings**. We let P_i be the cumulative probability of s_i, which is calculated as $P_i = \sum_{j=1}^{i} p_j$. Using these cumulative probabilities, we avoid recalculating $q_j = \sum_{s_i \in \mathcal{S}_j} p_i$ at each step of the algorithm. The way in which this algorithm generates all possible consecutive partitionings of $\mathcal{S}$ is based on the proof of Theorem 4.3. It proceeds by enumerating the $r-1$ valid partitioning indices, $i_1, \ldots, i_{r-1}$. To achieve this, $r-1$ nested **for** loops are executed, where the jth **for** loop proceeds from the previous index, κ_{j-1}, to κ, where $m = r + \kappa(r-1)$.

Algorithm 11 All_Consecutive_Partitionings

Input: The source alphabet and probabilities, $\mathcal{S}$ and $\mathcal{P}$. The code alphabet, $\mathcal{A}$. The sum of probabilities of $\mathcal{P}$, q.

Output: The valid partitioning points for an optimal partitioning, $\{i_1, \ldots, i_{r-1}\}_{\min}$

Method:

```
  P_0 ← 0; κ ← (m−r)/(r−1); d_min ← MAX_VALUE
  for i ← 1 to m do
    P_i ← P_{i−1} + p_i
  endfor
  for κ_1 ← 0 to κ do
    i_1 ← κ_1(r − 1) + 1
    for κ_2 ← κ_1 to κ do
      i_2 ← κ_2(r − 1) + 2
          ⋮
      for κ_{r−1} ← κ_{r−2} to κ do
        i_{r−1} ← κ_{r−1}(r − 1) + (r − 1); d ← 0
        for j ← 1 to r do
          d ← d + |P_{i_j} − P_{i_{j−1}} − q/r|
        endfor
        if d < d_min then
          d_min ← d; {i_1, ..., i_{r−1}}_min ← {i_1, ..., i_{r−1}}
        endif
      endfor
    endfor
  endfor
end Algorithm All_Consecutive_Partitionings
```

This algorithm enumerates all possible consecutive partitionings. This is stated and proved in the theorem given below.

Theorem 4.4. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, and let $\mathcal{A} = \{a_1, \ldots, a_r\}$ be the code alphabet, where $m = r+\kappa(r-1)$ for some integer $\kappa \geq 0$. Algorithm **All_Consecutive_Partitionings** enumerates all possible consecutive partitionings.

Proof. The theorem follows by a direct consequence of enumerating the possible partitions as detailed in the proof of Theorem 4.3. It is not included here to avoid repetition. □

Although Algorithm **All_Consecutive_Partitionings** enumerates and searches the entire space of all possible consecutive partitionings, its worst-case time complexity is bounded by a polynomial whose degree is given by the number of code alphabet symbols. This fact is stated and proved in the following theorem.

Theorem 4.5. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, and let $\mathcal{A} = \{a_1, \ldots, a_r\}$ be the code alphabet, where $m = r + \kappa(r-1)$ for some integer $\kappa \geq 0$. The worst-case time complexity of Algorithm **All_Consecutive_Partitionings** is:

(*i*) $O(m^{r-1})$ if $r < m$, and

(*ii*) $O(m)$ if $r = m$.

Proof. From the first $r-1$ **for** loops, we see that the jth **for** loop is executed from κ_{j-1} up to κ. This implies that $\kappa_1 \leq \kappa_2 \leq \ldots \leq \kappa_{r-1}$.

Since the first $r-1$ **for** loops are executed up to κ, and $\kappa_1 \leq \kappa_2 \leq \ldots \leq \kappa_{r-1}$, the total number of times the last **for** loop is executed is given by the number of ways of selecting a set of integers, $\{\kappa_1, \kappa_2, \ldots, \kappa_{r-1}\}$, from a set of $\kappa+1$ elements, $\{0, 1, \ldots, \kappa\}$, where $\kappa_1 \leq \kappa_2 \leq \ldots \leq \kappa_{r-1}$, where repetitions are allowed.

Using the result of Theorem 4.3, this corresponds to the number of $(r-1)$-combinations from a set of $\kappa + 1$ elements, where repetitions are allowed. This number is given by:

$$\binom{\kappa+r-1}{r-1} = \frac{(\kappa+r-1)!}{\kappa!(r-1)!}. \tag{4.8}$$

Since the last **for** loop involves r steps, the total number of steps required by Algorithm **All_Consecutive_Partitionings** is given by:

$$r\frac{(\kappa+r-1)!}{\kappa!(r-1)!} = r\ \frac{(\kappa+r-1)(\kappa+r-2)\ldots(\kappa+1)\kappa!}{\kappa!(r-1)!} \tag{4.9}$$

$$= \frac{A_0 + A_1\kappa + \ldots + A_{r-2}\kappa^{r-2} + A_{r-1}\kappa^{r-1}}{(r-1)!}, \tag{4.10}$$

where $A_0 = r(r-1)!$, $A_{r-1} = r$, $A_i = f_i(r)$ for $i = 1, \ldots, r-2$, and $f_i(.)$ is some well-defined function of r, but which does not contain m.

Equation (4.10) can also be written as follows:

$$r + \frac{A_1}{(r-2)!}\kappa + \ldots + \frac{A_{r-2}}{(r-2)!}\kappa^{r-2} + \frac{1}{(r-2)!}\kappa^{r-1}. \tag{4.11}$$

Since $m = r + \kappa(r-1)$, we have $\kappa = \frac{m-r}{r-1}$. Substituting κ for $\frac{m-r}{r-1}$ in (4.11), we have:

$$r + \frac{A_1}{(r-2)!}\frac{m-r}{r-1} + \ldots + \frac{A_{r-2}}{(r-2)!}\left(\frac{m-r}{r-1}\right)^{r-2} + \frac{1}{(r-2)!}\left(\frac{m-r}{r-1}\right)^{r-1}. \tag{4.12}$$

After expanding all the terms, $\left(\frac{m-r}{r-1}\right)^i$, for $i = 1, \ldots, r-1$, the highest exponent of m is of order $r-1$, which results from expanding $\left(\frac{m-r}{r-1}\right)^{r-1}$. Then, (4.12) can be written as follows:

$$B_0 + B_1 m + \ldots + B_{r-2}m^{r-2} + B_{r-1}m^{r-1}, \tag{4.13}$$

where $B_i = g_i(r)$, for $i = 0, \ldots, r-1$, and $g_i(.)$ is a well-defined function of r but not of m, and in particular, $B_{r-1} = \frac{1}{(r-2)!(r-1)^{r-1}}$. Indeed, for a fixed value of r, B_{r-1} is a constant, and hence (4.13) reduces to $O(m^{r-1})$. This proves Result (i).

We now consider the case when $m = r$ (or equivalently when $\kappa = 0$). Substituting κ for 0 and m for r in (4.8), we have:

$$m \begin{pmatrix} m-1 \\ m-1 \end{pmatrix} = m = O(m),$$

and (ii) is satisfied.

Hence the theorem. □

In order to implement the greedy adaptive Fano coding for multi-symbol code alphabets, the optimal consecutive partitioning has to be performed in each sublist in which the source symbol being encoded, $x[k]$, is contained. The number of partitioning steps for a single symbol is given by the length of the code word sent to the output. In the worst case, this number can take values up to $m-1$. Hence the worst-case time complexity of encoding a source symbol is $O(m^r)$. Therefore, this procedure is quite inefficient for large code alphabets (even for $r = 3$ or 4). To overcome this situation, we propose a greedy but accurate, linear-time algorithm that finds a sub-optimal partitioning.

4.6.2 Sub-Optimal Nearly-Equal-Probability Partitioning

Since the brute force algorithm that we presented in the previous subsection is quite inefficient, we now present a non-exhaustive, greedy algorithm that finds a *sub-optimal* consecutive partitioning. The implementation of this procedure is depicted in Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning**. Using this partitioning procedure, a source symbol can be encoded in $O(m)$ time. The source alphabet, the probabilities of occurrence and the code alphabet are as usual, and $x[k]$ is the symbol being encoded at time 'k'. The current range, $[t, b]$, contains the boundaries of $\mathcal{S}$ (or eventually the sublist) in which $x[k]$ is contained. After the partitioning procedure

Algorithm 12 Greedy_r-ary_Adaptive_Fano_Partitioning

Input: The source alphabet and probabilities, $\mathcal{S}$ and $\mathcal{P}$. The code alphabet, $\mathcal{A}$.
The current input symbol, $x[k]$. The current range, $[t, b]$.
The sum of probabilities of $\mathcal{P}$, q.

Output: The new range, $[t, b]$. The output symbol, $y[j]$.
The sum of probabilities of the new $\mathcal{P}$, q .

Method:

```
procedure Partition(S, P, A : list; x[k] : symbol; var t, b : integer; var q : real);
  i ← t; found ← false; p_temp ← 0; r ← |A|
  while not(found) do
    if s_i = x[k] then found ← true endif
    p_temp ← p_temp + p_i; i ← i + 1
    while p_temp < q/r do
      p_temp1 ← 0
      for j ← 1 to r − 1 do
        p_temp1 ← p_temp1 + p_i; i ← i + 1
        if s_i = x[k] then found ← true endif
      endfor
      p_temp ← p_temp + p_temp1
    endwhile
    if q/r − p_temp + p_temp1/2 < 0 then
      i ← i − r + 1; p_temp ← p_temp − p_temp1
    endif
    if found then
      b ← i − 1; q ← p_temp
    else
      t ← i; q ← q − p_temp
      if r = 1 then
        found ← true
      else
        r ← r − 1; p_temp ← 0
      endif
    endif
  endwhile
  return a_{|A|−r+1}
endprocedure
```

end Algorithm Greedy_r-ary_Adaptive_Fano_Partitioning

is performed, these boundaries represent the sub-sublist in which $x[k]$ is contained. The variable q stands for the sum of probabilities of those symbols between t and b. After performing the partitioning procedure, q contains the sum of probabilities of the resulting sub-sublist.

Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning** proceeds by partitioning the list into two sublists. The variable p_{temp} keeps track of the cumulative probability. The while loop is executed until p_{temp} is greater than or equal to $\frac{q}{r}$. If these two sublists do not satisfy the nearly-equal-probability property, the first $r-1$ elements of the second list have to be moved to the first sublist. In fact, if $\frac{q}{r} - p_{\text{temp}} + \frac{p_{\text{temp1}}}{2} < 0$, p_{temp1} (the last $r-1$ components added to p_{temp}) have to be moved to the first sublist.

If the symbol being encoded is found in the first sublist, the iteration stops, and a code alphabet symbol is returned. If it is not found, unity is subtracted from r, and the second sublist is partitioned in the same fashion. The labeling strategy used here consists of assigning a_1 to the first sublist, a_2 to the second sublist, and so on.

Formal algorithms for encoding/decoding are exactly the same as the formal corresponding Fano schemes for the r-ary case found in any textbook [58], except that in this case, the partitioning is as determined by Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning**. These algorithms are straightforward, and so not included here.

To conclude this chapter, we provide the analysis for the correctness of the greedy adaptive Fano coding for multi-symbol code alphabets using the sub-optimal, linear-time partitioning algorithm presented in this section. We shall show that given any source sequence, the output generated by encoding using the partitioning procedure given in Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning** has enough information to reproduce the original sequence by performing the reverse process, decoding, using the same partitioning procedure. We also show that this model for multi-symbol code alphabets achieves on-line compression. The two theorems and their corresponding proofs are given below.

Theorem 4.6. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, and the code alphabet $\mathcal{A} = \{a_1, \ldots, a_r\}$. Suppose that the input sequence $\mathcal{X} = x[1] \ldots x[M]$ is encoded by invoking the partitioning procedure of Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning** yielding an output sequence $\mathcal{Y} = y[1] \ldots y[R]$. If $\mathcal{Y}$ is decoded by

invoking the same partitioning procedure into $\mathcal{X}^* = x^*[1] \ldots x^*[T]$, then $\mathcal{X}^* = \mathcal{X}$ (i.e. $T = M$, and $x^*[i] = x[i]$, $\forall i$, $i = 1, \ldots, M$).

Proof. The proof of this theorem is similar, but a generalization, of that of Theorem 4.1. It can be achieved by a two-level induction : one for each encoding step 'k', and at the second level, another inductive argument (for each k) on the number of partitioning steps 'j'.

The main difference between the two proofs is that in the multi-symbol case, after a partitioning is performed, the two "extremes" of the resulting sublist, t and b, are, in most cases, both changed. The proof is completed by showing that, by using the respective algorithms, these values of t and b are the same for both the encoder and the decoder for *every* partitioning step 'j'. We then argue that since both the encoder and the decoder maintain the same values of t, b and p, their partitioning sublists are always identical, and so they together achieve lossless compression and decompression. The complete proof is not included here in order to avoid repetition. □

Theorem 4.7. Under the conditions of Theorem 4.6, Algorithm **Greedy_r-ary_Adaptive_Fano_Partitioning** achieves *on-line compression and decompression.*

Proof. The proof of this theorem is achieved by generalizing the arguments of the proof of Theorem 4.2. The arguments are, indeed, very similar and so omitted here to avoid repetition. □

4.7 Conclusions

In this chapter, we presented a "greedy" adaptive version of Fano's method. To the best of our knowledge, it has not been proposed in the literature yet. In the initial part of this chapter, we developed the encoding algorithm for the multi-symbol source alphabet and the binary code alphabet. For this scenario, we also developed the partitioning procedure and the decoding algorithm.

This chapter includes a formal proof for the correctness of the encoding and the decoding algorithms. We have shown that given a source sequence, $\mathcal{X}$, which is encoded by using Algorithm **Greedy_Adaptive_Fano_Encoding** generating an encoded

sequence, $\mathcal{Y}$, if $\mathcal{Y}$ is decoded using Algorithm **Greedy_Adaptive_Fano_Decoding**, the original source sequence $\mathcal{X}$ is recovered. This encoding is done in an *on-line* manner, which is very useful in communications and networking applications, such as the internet, ftp, e-commerce, e-mail, audio and video streaming, etc.

With regard to complexity, the worst-case time complexity of the adaptive Fano encoding algorithm for the binary code alphabet has been shown to be $O(mM)$, for an input sequence of size M, and m source alphabet symbols. For the adaptive Fano decoding algorithm, the worst-case complexity is $O(mM)$, where M is the size of the original sequence. Although the space complexity for the greedy adaptive Fano coding is the same as that of the adaptive Huffman coding, in practice, the latter requires about six times as much space as that of the former. This implies that the greedy adaptive Fano coding is quite advantageous in practical applications, specially, when memory constraints are tight.

In order to test the efficiency of the adaptive Fano coding algorithms, we conducted various experiments on files of the Calgary corpus and the Canterbury corpus. Our results demonstrate that the greedy adaptive Fano method compresses marginally less than the adaptive Huffman scheme. In terms of speed, the former is faster in the compression phase and slower in the decompression phase.

We also extended the greedy adaptive version of Fano's method to multi-symbol code alphabets. We have studied the problem of obtaining consecutive partitionings that satisfy the principles of Fano coding. To find the optimal partitioning, we have proposed a brute force algorithm that searches the entire space of *all* possible partitionings. We have shown that this algorithm operates in $O(m^{r-1})$ time. As opposed to this, we also proposed a greedy algorithm that yields a sub-optimal, but accurate, consecutive partitioning. Using this procedure, a single source symbol can be encoded in linear time. The corresponding results for the correctness, and the on-line compression achieved by using this procedure have been stated. The problem of devising a non-brute force algorithm that finds the optimal consecutive partitioning remains open, and constitutes a possible avenue for future extensions of the work that we have done in this chapter.

Chapter 5

Adaptive Fano Coding Using Binary Search Trees

Binary Search Trees (BST) have been used in a wide range of applications in storage, including dictionaries, symbol tables, etc. A BST is a tree in which a sorted list containing its records can be obtained by traversing the tree in a symmetric order. BSTs can be maintained statically when the statistical information about the number of accesses to the records is known *a priori*. When the probabilistic distribution about the records is unknown, adaptive schemes are the most appropriate ones. These schemes, update the BST during the search process.

In this chapter, we present an adaptive Fano coding approach that uses BSTs. We introduce a novel heuristic that adaptively updates the Fano tree during the encoding phase. The corresponding equivalent procedure is performed in the decoding phase. We also introduce new tree-based operators, namely the *Shift-To-Left* operator and the *Shift-To-Right* operator, which are performed whenever the tree becomes unbalanced from the Fano coding perspective. The decision on applying these operators is based on the heuristic that we introduce. Our algorithm performs very well in stationary sources, for which we prove that asymptotically no change is needed after the transient behavior finishes. We present the corresponding theoretical analysis for this scenario, as well as the formal theory for the tree operators and the heuristic alluded to above. Finally, we present the empirical results that we have obtained from running our algorithms

on files of the Calgary corpus and the Canterbury corpus.

5.1 Binary Search Trees

Binary Search Trees (BSTs) have been used in a variety of applications, including symbol tables, dictionaries, etc. In this research work, we use them in adaptive Fano coding.

Consider a set of records whose keys are given by the ordered set of distinct elements $\mathcal{K} = \{k_1, \ldots, k_m\}$, where $k_1 < \ldots < k_m$. Let $\mathcal{Q} = [q_1, \ldots, q_m]$ be the access probabilities of $\{k_1, \ldots, k_m\}$ respectively. A BST is a tree in which every node has zero, one or two children, and satisfies the following property:

Given a BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, each node contains an element of $\mathcal{K}$ in such a way that for every internal node, t_i,

$$k_{L_i} < k_i < k_{R_i}, \tag{5.1}$$

where L_i and R_i stand for the left child and the right child of t_i respectively.

Searching for a particular record in a BST involves traversing the tree, starting from the root node. By going in the direction to the leaves, the value of each node is compared to the key of the record being searched. We go to the left child if the key being searched is less than the value of the current node, and we go to the right child if the key is greater than the value of the current node. If the key being searched is equal to that of the current node, the search stops, and the node found is returned.

If the key is not found after a leaf is reached, an *insertion* operation must be performed so that the new record is introduced into the BST. Unless otherwise stated, we assume that all the record keys, $\{k_1, \ldots, k_m\}$, are already in the BST, as will be the case in our application domain.

By following the procedure given in [74], the optimal BST can be constructed using dynamic programming in $O(m^2)$ time and space. Alternatively, using dynamic programming and divide-and-conquer techniques [161], a nearly-optimal BST can be

constructed in $O(m)$ space and $O(m \log m)$ time. These two approaches can be used whenever the statistical information about the access to the records is known beforehand.

As opposed to this, we assume that these probabilities are unknown, and the structure and the content of the BST are dynamically changed while the records are searched for, in the tree.

5.1.1 Adaptive Structuring of BSTs

In this section, we present an overview of the most well-known BST structuring approaches proposed to maintaining adaptive binary search trees. The *move-to-root* heuristic was proposed by Allen and Munro [6]. This approach is very simple; each time a record is accessed, it is moved up to become the root of the tree using rotation operations. The aim of this approach is to maintain the most frequently accessed records near the root, and consequently, to minimize the average cost of searching.

Another approach, the *simple exchange rule*, was also introduced by Allen and Munro [6]. It consists of rotating the accessed record one level upwards the root. The aim of this simple approach is to keep the most-frequently accessed nodes near the root and the least accessed nodes near the leaves of the tree. Although this approach is not very efficient, it has the advantage that it does not use extra space.

Apart form these structuring techniques, we mention the technique due to Sleator and Tarjan [150] known as *splaying*, who also use their own tree structure called the *splay tree*. The main idea of this technique is to move the accessed record towards the root, and to simultaneously allow accesses to each record by an in-order traversal of the tree. The splaying tree structuring techniques has reported good results even for highly time-variant access probabilities.

Another scheme found in the literature is known as *monotonic trees* [16]. The static version of this scheme maintains the BST arranged in such a way that the most accessed record is the root of the tree and the sub-trees rooted at the root's children are recursively arranged in the same manner. In the adaptive version of this scheme, each record maintains an extra location used to count the number of times it was

accessed. The tree is restructured as follows: when a record is accessed, its counter is incremented. This record is then rotated upwards until its parent has a higher frequency than itself or it becomes the root of the tree. This approach performs poorly for key sets with high entropy. Empirical results have also shown that, on the average, it behaves poorly.

Other adaptive binary search tree approaches are *biasing* [14], *dynamic binary search* [94], *weighted randomization* [8], *deepsplaying* [148], and the technique that uses *conditional rotation* [23]. In the next section, we discuss the latter in detail, since it forms the foundation for this research work involving adaptive Fano coding. The other schemes are not described here in the interest of brevity. They are, indeed, irrelevant to our present study.

5.2 The Use of Conditional Rotations in BSTs

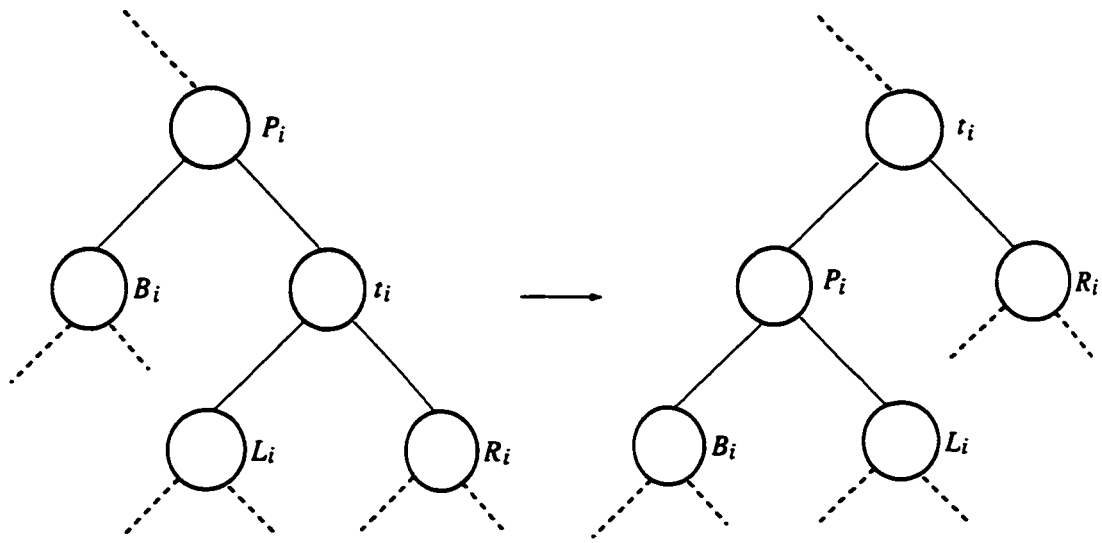
The use of conditional rotations in adaptive BSTs was introduced in 1993 by Cheetham, Oommen, and Ng [23]. The basic idea of this approach is to maintain certain key pieces of information in each node. These are used by the heuristic called *conditional rotation.* Since this approach forms the basis for developing our adaptive Fano coding schemes using BSTs, we discuss it in detail below.

5.2.1 Rotation Operations

The *rotation operation* (also known as the *promotion operation* [79]) was introduced by Adel'son-Vel'skii and Landis [4]. This operation is performed on an internal node in such a way that, among other changes, this node and its parent interchange roles. Prior to introducing an example that helps to understand how this operator works, we introduce the relevant notation used in the rotation operation.

Consider a BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. If t_i is any internal node of $\mathcal{T}$, then:

- P_i is the parent of t_i,
- L_i is the left child of t_i,



(a) The BST before the rotation. (b) The BST after the rotation.

Figure 5.1: A rotation operation performed on node t_i.

- R_i is the right child of t_i, and
- B_i is the brother (or sibling) of t_i.

Using these primitives, B_i can also be defined as follows:

- L_{P_i} if t_i is the right child of P_i, and
- R_{P_i} if t_i is the left child of P_i.

Example 5.1. Let $\mathcal{T}$ be the BST depicted in Figure 5.1(a). Suppose that we want to perform a rotation on t_i. The BST after the rotation is depicted in Figure 5.1(b). Observe that t_i becomes the parent of its previous parent, P_i, and R_i remains unchanged. L_i becomes the right child of P_i, and after the rotation, is the brother of B_i. The rest of the tree remains unchanged, i.e. below R_i, L_i and B_i, and above P_i. □

The properties of the rotation operation can be found in [4, 23].

5.2.2 Conditional Rotation Heuristics

We now discuss the conditional rotation heuristic introduced in [23]. This heuristic requires three extra memory locations for each node which represent the number of accesses to the record, the number of accesses to the subtree rooted at that record, and the Weighted Path Length (WPL) of the subtree rooted at that record.

The aim of this approach is to minimize the WPL of the subtree rooted at t_i, where the node t_i contains the record being accessed. The fields that contain the information about the number of accesses to t_i, the number of accesses to the subtree rooted at t_i, etc., are updated, and a rotation on t_i is performed whenever the WPL decreases as a result of the rotation. Since the WPL of the entire tree depends also on that of t_i, this results in a decrease of the WPL of the *entire* tree.

In the interest of formality, we introduce the notation used in the conditional rotation heuristic:

$\alpha_i(n)$ is the total number of accesses to node t_i up to time n.

$\tau_i(n)$ is the total number of accesses to $\mathcal{T}_i$, the subtree rooted at t_i, up to time n, and is calculated as follows:

$$\tau_i(n) = \sum_{t_j \in \mathcal{T}_i} \alpha_j(n) \,. \tag{5.2}$$

$\kappa_i(n)$ is the WPL of $\mathcal{T}_i$, the subtree rooted at t_i, at time n, and is calculated as follows:

$$\kappa_i(n) = \sum_{t_j \in \mathcal{T}_i} \alpha_j(n)\lambda_j(n) \,, \tag{5.3}$$

where $\lambda_j(n)$ is the path length from t_j up to node t_i.

By using simple induction, it can be shown that:

$$\kappa_i(n) = \sum_{t_j \in \mathcal{T}_i} \tau_j(n) . \tag{5.4}$$

The values of $\tau_i(n)$ and $\kappa_i(n)$ can also be calculated as follows:

$$\tau_i(n) = \alpha_i(n) + \tau_{L_i}(n) + \tau_{R_i}(n), \text{ and} \tag{5.5}$$

$$\kappa_i(n) = \tau_i(n) + \kappa_{L_i}(n) + \kappa_{R_i}(n) . \tag{5.6}$$

These values are updated for all the nodes in the path, starting from the root until the record being searched is found.

The criteria for performing rotations is based on the aim of decreasing the WPL of the entire tree as a result of the rotation. Let α', τ' and κ' be the corresponding values after a rotation is performed. Then, a rotation is performed on node t_i if the following condition is satisfied:

$$\theta_i(n) = \kappa_{P_i}(n) - \kappa'_i(n) > 0 . \tag{5.7}$$

Observe that $\kappa'_i(n)$ is the value that the WPL would have attained *if* a rotation is performed on t_i. Clearly, a rotation is performed whenever $\kappa'_i(n) < \kappa_{P_i}(n)$. This implies that the WPL of t_i will decrease, and consequently, it can be shown that the WPL of the entire tree will decrease too.

The approach discussed above is called the *naive* conditional rotation heuristic. An *optimized* heuristic was also proposed in [23], which consists of maintaining, for each node, the value of τ only. The rotation operation is performed on node t_i if $\psi_i \geq 0$, where ψ_i is defined as follows:

$$\psi_i(n) = 2\tau_i(n) - \tau_{R_i}(n) - \tau_{P_i}(n), \text{ if } t_i \text{ is a left child, and}$$
$$\psi_i(n) = 2\tau_i(n) - \tau_{L_i}(n) - \tau_{P_i}(n), \text{ if } t_i \text{ is a right child.}$$

The procedure for searching and updating are performed at the same time. Starting from the root towards the leaves, the key being searched, k_i, is compared to that of the current node, k_j. If they are distinct, $\tau_j(n)$ is updated, and k_i is searched in the subtree rooted at t_{L_j} if $k_i < k_j$, or in the subtree rooted at t_{R_j} if $k_i > k_j$. When k_i is found, $\tau_i(n)$ is updated and a rotation is performed on node t_i whenever $\psi_i(n) \geq 0$. Note that in this approach, it is assumed that k_i is always in $\mathcal{T}$.

It has been shown that the naive conditional rotation heuristic and its optimized version asymptotically achieve a BST whose access cost is arbitrarily close to the optimal. Empirical results have also shown that these heuristics achieve almost-optimal costs on typical probability distributions, and are always superior to splay trees, move-to-root trees, monotonic trees, and various other adaptive BST schemes.

5.3 Adaptive Fano Coding Using BSTs

We now present an alternative solution for the adaptive Fano coding using BSTs. We introduce a new structure that we called the *Fano BST*, which is a particular case of a BST, and *shift operators* which can be performed on any BST. We also present the encoding algorithm and the corresponding decoding algorithm, which use the conditional shifting heuristic introduced here. Although it has its foundations on the conditional BST, the heuristic that we use is completely different from the latter. More precisely, our heuristic is based on the principles of the Fano coding, as opposed to minimizing the access cost.

5.3.1 Motivation

The adaptive Fano coding involves the maintenance of a Fano tree, which is used to adaptively encode an input sequence. We assume that the probabilities of the source

symbols are unknown. At the beginning of the encoding process, the uncertainty about the occurrence of the next symbol, will cause many statistical and structural changes in the tree. In learning paradigms, this is known as the *transient behavior*. After some time, and under the assumption of the *stationarity* of the source, fewer changes are expected, and hence the structural changes are dramatically reduced so as to achieve the desired behavior, namely, the computation of the optimal Fano encoding and decoding mechanisms.

5.3.2 Fano BSTs

As mentioned earlier, the basis for the particular "species" of binary search trees introduced here, the Fano BST, comes from the structure used in the conditional rotation heuristic. Broadly speaking, a Fano BST is a BST in which the number of accesses of all the internal nodes are set to zero, and the number of accesses of each leaf represents the number of times that the symbol associated with that leaf has appeared so far in the input sequence. The aim is to maintain the tree balanced in such a way that for every internal node, the weight of the left child is as nearly-equal as possible to that of the right child. Formal definitions of the partitioning BST and the Fano BST are given below. In order to simplify the notation, we let α_i, τ_i, and κ_i be the corresponding values (as defined in the conditional rotation heuristic) contained in node t_i at time n, i.e. $\alpha_i(n)$, $\tau_i(n)$, and $\kappa_i(n)$ respectively.

Definition 5.1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq \ldots \geq p_m$, and the code alphabet $\mathcal{A} = \{0, 1\}$. A *partitioning BST* is a binary tree, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, whose nodes are identified by their indices (for convenience, also used as the keys $\{k_i\}$), and whose fields are the corresponding values of τ_i. Furthermore, every partitioning BST satisfies:

(*i*) Each node t_{2i-1} is a leaf, for $i = 1, \ldots, m$, where s_i is the ith alphabet symbol satisfying $p_i \geq p_j$ if $i < j$.

(*ii*) Each node t_{2i} is an internal node, for $i = 1, \ldots, m-1$.

Remark 5.1. Given a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, the number of accesses to a leaf node, α_{2i-1}, is a counter, and p_i refers to either α_{2i-1} (the frequency counter) or the probability of occurrence of the symbol associated with t_{2i-1}. In fact, the probability of occurrence of s_i can be estimated as follows:

$$p_i = \frac{\alpha_{2i-1}}{\sum_{j=1}^{m} \alpha_{2j-1}}. \tag{5.8}$$

We now introduce a particular case of the partitioning BST, the *Fano BST*. This tree has the added property that each partitioning step is performed by following the principles of the Fano coding, i.e. the weights of the two new nodes are as nearly equal as possible. This is formally defined below.

Definition 5.2. Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a partitioning BST. $\mathcal{T}$ is a *Fano BST*, if for every internal node, t_{2i}, the following conditions are satisfied:

(a) $\tau_{R_i} - \tau_{L_i} \leq \tau_{2i+1}$ if $\tau_{L_i} < \tau_{R_i}$,

(b) $\tau_{L_i} - \tau_{R_i} \leq \tau_{2i-1}$ if $\tau_{L_i} > \tau_{R_i}$, or

(c) $\tau_{L_i} = \tau_{R_i}$.

The procedure for constructing a Fano BST from the source alphabet symbols and their probabilities of occurrence is depicted in Algorithm **Fano_BST_Construction**. The partitioning procedure is similar to that of the greedy adaptive Fano coding presented in Chapter 4. Each time a partitioning is performed, two sublists are obtained[1], and two new nodes are created, t_{n_0} and t_{n_1}, which are assigned to the left child and the right child of the current node, t_n. This partitioning is recursively performed until a sublist with a single symbol is obtained. We use the subindex n to refer to the fields of node t_n. For example, τ_n represents the total number of accesses to node t_n.

[1]Each time the procedure FanoBST(...) is invoked, two sublists of $\mathcal{S}$ and $\mathcal{P}$, respectively, are sent as parameters. These sublists are specified by a *lower-bound* index and an *upper-bound* index, u and l respectively.

Algorithm 13 Fano_BST_Construction

Input: The source alphabet and probabilities, $\mathcal{S}$ and $\mathcal{P}$.
Output: The Fano BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$.
Method:

```
procedure FanoBST(S, P : list; u, l : integer; t_n : node; τ_n : real);
  i ← u − 1; p ← 0
  while p < τ_n/2 do
    i ← i + 1; p ← p + p_i
  endwhile
  if τ_n/2 − p + p_i/2 < 0 then    // Are the properties of Definition 5.2 satisfied?
    i ← i − 1; p ← p − p_i
  endif
  k_n ← 2i
  Create nodes t_n0 and t_n1
  L_n ← t_n0; R_n ← t_n1
  if u < i then
    τ_Ln ← p; FanoBST(S, P, u, i, L_n, τ_Ln)
  else
    α_Ln ← p_t; σ_Ln ← s_t; k_Ln ← 2i − 1
  endif
  if l > i + 1 then
    τ_Rn ← τ_n − p; FanoBST(S, P, i + 1, l, R_n, τ_Rn)
  else
    α_Rn ← p_b; σ_Rn ← s_b; k_Rn ← 2i + 1
  endif
endprocedure
procedure FanoBSTSort(t_n : node; var T : list; var m : integer);
  if L_n ≠ NIL then
    FanoBSTSort(L_n, T, m)
  endif
  m ← m + 1; t_i ← t_n
  if R_n ≠ NIL then
    FanoBSTSort(R_n, T, m)
  endif
endprocedure
Create node root
τ_root ← Σ_{i=1}^{m} p_i
FanoBST(S, P, 1, m, root, τ_root)
T ← {}; m ← 0; FanoBSTSort(root, T, m)
end Algorithm Fano_BST_Construction
```

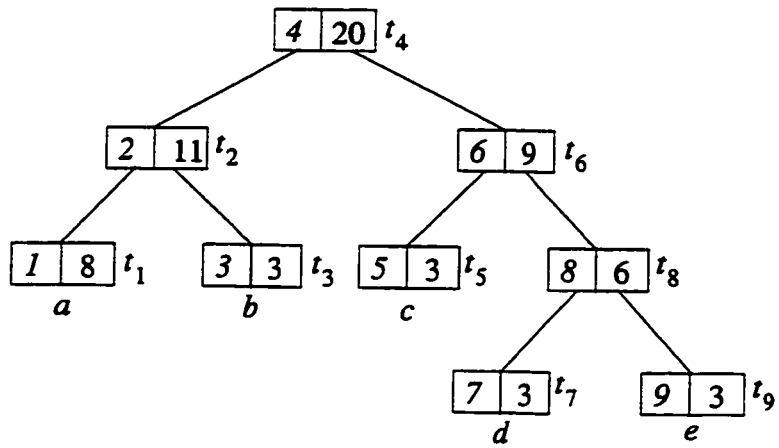


Figure 5.2: An example of a Fano BST constructed from $\mathcal{S} = \{a, b, c, d, e\}$ and $\mathcal{P} = [8, 3, 3, 3, 3]$.

In order to ensure that $\mathcal{T}$ satisfies properties (i) and (ii) of Definition 5.1, and also the conditions of Definition 5.2, the Fano BST generated by procedure FanoBST(...) must be rearranged as if each node were accessed in a traversal order, from left to right. The sorted Fano BST is generated by invoking procedure FanoBSTSort(...), which produces a list of nodes in the desired order, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$.

We now present an example that helps to clarify the procedures given in Algorithm **Fano_BST_Construction**.

Example 5.2. Consider the source alphabet $\mathcal{S} = \{a, b, c, d, e\}$ whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3]$, and the code alphabet $\mathcal{A} = \{0, 1\}$. A partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_9\}$, constructed with $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{A}$ is depicted in Figure 5.2. This partitioning BST is also a Fano BST, i.e. it also satisfies property of Definition 5.2 for every internal node, t_{2i}, for $i = 1, \ldots, m-1$. The corresponding Fano BST construction procedure is depicted in Figure 5.3.

The internal nodes are t_2, t_4, t_6 and t_8, i.e. t_{2i} for $i = 1, \ldots, 4$. The leaves are the nodes t_1, t_3, t_5, t_7, and t_9. For every node, the cell on the left contains the key, and the cell on the right contains the total number of accesses to the subtree rooted at that node. Each leaf is also associated with a source alphabet symbol, s_i, where $2i - 1$ is the key of that leaf.

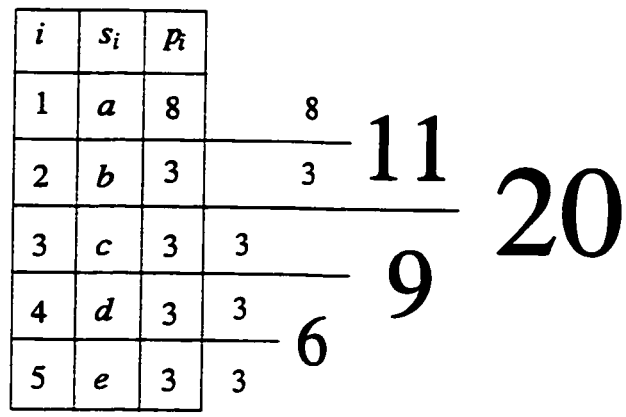


Figure 5.3: Partitioning BST construction procedure for $\mathcal{S} = \{a, b, c, d, e\}$, $\mathcal{P} = [8, 3, 3, 3, 3]$, and $\mathcal{A} = \{0, 1\}$. The resulting tree is a Fano BST.

All the nodes of $\mathcal{T}$ are sorted from left to right, in an ascending order of key, from 1 to 9. The leaves are also sorted from left to right in a descending order of the frequency counter.

For each internal node, the key contains the value $2i$, where i represents the index of the last source symbol of the top-most list derived from the partitioning. For example, the first partitioning produces $\{a, b\}$ and $\{c, d, e\}$ whose weights are 11 and 9 respectively. The node at which the partitioning is done is the root. Since the last symbol of $\{a, b\}$ is b whose index in $\mathcal{S}$ is $i = 2$, the key for the root is $2i = 4$. As we will see later, the correspondence between the index of the source symbol and the key of the internal node is very useful in the implementation of the encoding algorithm. □

Remark 5.2. The structure of the Fano BST is similar to the structure of the BST used in the conditional rotation heuristic introduced in [23]. The difference, however, is that since every internal node does not represent an alphabet symbol, the values of α_{2i} are all 0, and the quantities for the leaf nodes, $\{\alpha_{2i-1}\}$, are set to $\{p_i\}$ or to frequency counters representing them.

Clearly, the total number of accesses to the subtree rooted at node t_{2i}, τ_{2i}, is obtained as the sum of the number of accesses to all the leaves of $\mathcal{T}_{2i}$. This is proven in the lemma below[2].

[2]The result is fairly straightforward, but included for the sake of completeness. Also, it is included so that the parallel between Fano BSTs and traditional BSTs becomes clear to the reader.

Lemma 5.1. Let $\mathcal{T}_{2i} = \{t_1, \ldots, t_{2s-1}\}$ be a subtree rooted at node t_{2i}. The total number of accesses to $\mathcal{T}_{2i}$ is given by:

$$\tau_{2i} = \sum_{j=1}^{s} \alpha_{2j-1} = \tau_{L_{2i}} + \tau_{R_{2i}} \,. \tag{5.9}$$

Proof. By using (5.2), the total number of accesses to the subtree rooted at t_{2i} is calculated as follows:

$$\tau_{2i} = \sum_{j=1}^{2s-1} \alpha_j \,. \tag{5.10}$$

The first equality holds because

$$\tau_{2i} = \sum_{j=1}^{s-1} \alpha_{2j} + \sum_{j=1}^{s} \alpha_{2j-1} = \sum_{j=1}^{s} \alpha_{2j-1} \,, \tag{5.11}$$

the last step being a consequence of the fact that $\alpha_{2j} = 0$ for all j.
The second equality of (5.9) uses Lemma 1 of [23], whence,

$$\tau_{2i} = \alpha_{2i} + \tau_{L_{2i}} + \tau_{R_{2i}} \,. \tag{5.12}$$

The result again follows by invoking the property $\alpha_{2i} = 0$, which is true for all i. The lemma is thus proved. □

We now present a result that relates the WPL of a Fano BST and the average code word length of the encoding schemes generated from that tree. For any Fano BST, $\mathcal{T}$, κ is calculated using (5.4). By optimizing on the relative properties of κ and τ, we can show that the average code word length of the encoding schemes generated from $\mathcal{T}$, $\bar{\ell}$, can be calculated from the values of κ and τ that are maintained at the root. Note

that this is done with a single access – without traversing the entire tree. This result is stated and proved in Theorem 5.1 given below.

This is quite an "intriguing" result[3]. The issue at stake is to compute the expected value of a random variable, in this case, the *expected code word length.* In general, this can be done if we are given the values that the random variable assumes, and their corresponding probabilities. The actual computation would involve the summation (or an integral in the case of continuous random variables) of the product of the values and their associated probabilities. Theorem 5.1 shows how this expected code word length can be computed quickly – without *explicitly* computing either the product or the summation. However, this is done *implicitly*, since κ and τ take these factors into consideration. Since the Fano BST is maintained adaptively, the average code word length is also maintained adaptively. Invoking this result, we can obtain the average code word length by a single access to the root of the Fano BST.

Theorem 5.1. Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a Fano BST constructed from the source alphabet $\mathcal{S} = \{s_1, \ldots s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$. If $\phi : \mathcal{S} \rightarrow \{w_1, \ldots, w_m\}$ is an encoding scheme generated from $\mathcal{T}$, then

$$\bar{\ell} = \sum_{i=1}^{m} p_i \ell_i = \frac{\kappa_{root}}{\tau_{root}} - 1, \tag{5.13}$$

where ℓ_i is the length of w_i, τ_{root} is the total number of accesses to $\mathcal{T}$, and κ_{root} is as defined in (5.3).

Proof. From Lemma 5.1, we know that $\tau_{root} = \sum_{j=1}^{m} \alpha_{2j-1}$. Since, from Remark 5.2, for every internal node, $\alpha_{2i} = 0$, we can write (5.3) for κ_{root} as follows:

[3]Note that in the actual design of the encoding methods using Fano BSTs, we do not keep the value of κ for each node. This is because the heuristic that we use to update the tree at each encoding step is based on the *nearly-equal-probability property* discussed presently, which only uses the values of τ to optimize the encoding. However, it is important to highlight the utility of this result, since other heuristics can be proposed which could additionally use the value of κ, and consequently, κ could have prominence in optimizing the expected code word length.

$$\kappa_{root} = \sum_{i=1}^{m-1} \alpha_{2i}\lambda_{2i} + \sum_{i=1}^{m} \alpha_{2i-1}\lambda_{2i-1} = \sum_{i=1}^{m} \alpha_{2i-1}\lambda_{2i-1}\,. \tag{5.14}$$

Consider the ratio $\frac{\kappa_{root}}{\tau_{root}}$:

$$\frac{\kappa_{root}}{\tau_{root}} = \sum_{i=1}^{m} \frac{\alpha_{2i-1}}{\sum_{j=1}^{m} \alpha_{2j-1}} \lambda_{2i-1}\,. \tag{5.15}$$

By invoking Remark 5.1 we have $p_i = \frac{\alpha_{2i-1}}{\sum_{j=1}^{m} \alpha_{2j-1}}$, and hence (5.15) can be written as follows:

$$\frac{\kappa_{root}}{\tau_{root}} = \sum_{i=1}^{m} p_i\lambda_{2i-1} = \sum_{i=1}^{m} p_i(\ell_i + 1) = \bar{\ell} + 1\,. \tag{5.16}$$

The last equality is a consequence of the fact that $\lambda_{2i-1} = \ell_i + 1$, which is true because λ_i involves counting the *nodes* along the path from the root to the leaf associated with s_i, and ℓ_i is obtained by counting the *edges* in the corresponding path. □

From Theorem 5.1, we see that the WPL and $\bar{\ell}$ are closely related. The smaller the WPL, the smaller the value of $\bar{\ell}$. Consequently, the problem of minimizing the WPL of a Fano BST is equivalent to minimizing the average code word length of the encoding schemes obtained from that tree.

5.3.3 Shifting Operations in Partitioning BSTs

The aim of our on-line encoding/decoding is to maintain a structure that maximally contains and utilizes the statistical information about the source. Using this structure, the current symbol is encoded and the structure is updated in such a way that the next symbol is expected to be encoded as optimally as possible. Various structure models have been proposed, including Markov models (higher-order models), dictionaries, lists,

Huffman trees, etc. The latter can be combined with the other models to achieve even more efficient compression.

Alternatively, we propose to use our new structure, namely the Fano BST defined in Section 5.3.2, which is adaptively maintained by simultaneously encoding and learning about the statistics of the source[4]. The learning process requires that two separate phases are sequentially performed. The first consists of updating the frequency counters of the current symbol, and the second involves changing the structure of the partitioning BST so as to maintain a Fano BST. Other adaptive encoding techniques, such as Huffman coding or arithmetic coding, utilize the same sequence of processes: encoding and then learning.

After the encoder updates the frequency counter of the current symbol and the corresponding nodes, the resulting partitioning BST may need to be changed so that a Fano BST is maintained. To achieve this, we introduce two new *shift* operators which can be performed on a partitioning BST : the *Shift-To-Left (STL)* operator and the *Shift-To-Right (STR)* operator. Broadly speaking, these operators consist of removing a node from one of the sublists obtained from the partitioning, and inserting it into the other sublist, in such a way that the new partitioning satisfies the nearly-equal-probability property. We discuss them in detail in the next two subsections.

5.3.3.1 The Shift-To-Left Operator

The STL operator performed on an internal node of a partitioning BST consists of removing the left-most leaf of the subtree rooted at the right child of that node, and inserting it as the right-most leaf in the subtree rooted at the left child of that node.

The relation between the STL operator and the Fano code construction procedure is the following. Suppose that a list $\mathcal{P} = [p_1, \ldots, p_m]$ has already been partitioned into two new sublists, $\mathcal{P}_0 = [p_1, \ldots, p_k]$ and $\mathcal{P}_1 = [p_{k+1}, \ldots, p_m]$. The equivalent to the STL operator for this scenario consists of deleting p_{k+1} from $\mathcal{P}_1$ and inserting it into the last position of $\mathcal{P}_0$, yielding $\mathcal{P}'_0 = [p_0, \ldots, p_k, p_{k+1}]$ and $\mathcal{P}'_1 = [p_{k+2}, \ldots, p_m]$.

[4]The structure that we introduce here, namely the Fano BST, could also be combined with other structure models, such as Markov models, dictionary-based compression, PPM schemes, etc., to achieve much more efficient compression. In this thesis, we use the zeroth-order model. The use of Fano BSTs with higher-order models is currently being investigated.

In order to introduce the formal procedure for the STL operator, we give below the assumptions under which this operator is performed.

Notation STL: Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, in which the weight of each node, t_l, is τ_l, and the key for each internal node is k_l, for $l = 1, \ldots, m$. Let

- t_i be an internal node of $\mathcal{T}$,
- R_i be also an internal node of $\mathcal{T}$,
- t_j be the left-most leaf of the subtree rooted at R_i, and
- t_k be the right-most leaf of the subtree rooted at L_i.

Using this notation, we can identify three mutually exclusive cases in which the STL operator can be applied. These cases are listed below, and the rules for performing the STL operation and the corresponding examples are discussed thereafter.

STL-1 : $P_{P_j} = t_i$ and L_i is a leaf.

STL-2 : $P_{P_j} \neq t_i$ and L_i is a leaf.

STL-3 : L_i is *not* a leaf.

The STL operator performed in the scenario of Case STL-1 is discussed below.

Rule 5.1 (STL-1). Consider a partitioning BST, $\mathcal{T}$, and the **Notation STL**. Suppose that the scenario is that of Case STL-1. The STL operator applied to the subtree rooted at node t_i consists of the following operations:

(a) the value $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) k_i and k_{P_j} are swapped[5],

[5]In the actual implementation, the Fano BST can be maintained in an array, in which the node t_l, $1 \leq l \leq 2m-1$, can be stored at position l. In this case, and in all the other cases of STL and STR, swapping these two keys could be avoided, and searching the node t_l could be done in a single access to position l in the array.

(c) B_j becomes the right child of t_i,

(d) P_j becomes the left child of t_i,

(e) t_k becomes the left child of P_j, and

(f) t_j becomes the right child of P_j.

□

Remark 5.3. The node on which the STL operator is applied, t_i, can be any internal node or the root satisfying the **Notation STL**. Example 5.3 given below will help to understand how the STL-1 operator works.

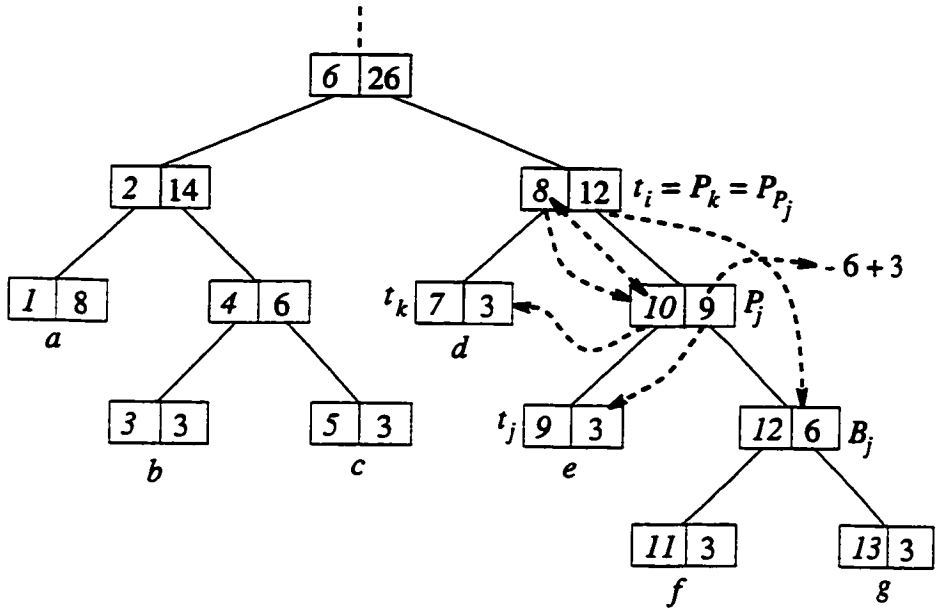
Example 5.3. Consider the source alphabet $\mathcal{S} = \{a, b, c, d, e, f, g\}$ whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$. Let $\mathcal{T}$ be the partitioning BST, constructed from $\mathcal{S}$ and $\mathcal{P}$, depicted in Figure 5.4(a). Suppose that we perform an STL operation on the subtree rooted at node t_i following Rule 5.1 (because we are in the scenario of Case STL-1). The resulting tree, $\mathcal{T}'$, is the partitioning BST depicted in Figure 5.4(b). The pointers in the new tree resulting from the STL operation correspond to the dashed arcs depicted in Figure 5.4(a). □

Observe in Figure 5.4(b) that the tree obtained after applying the STL-1 operator results in a partitioning BST. This is stated and proved for the operator, in general, in Lemma 5.2 given below.

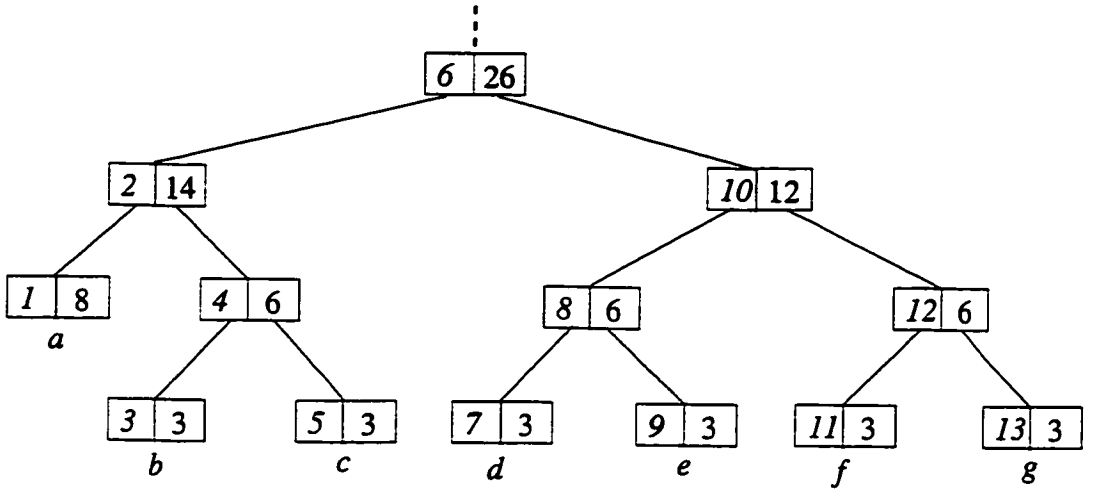
Lemma 5.2 (STL-1 validity). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, specified as per **Notation STL**. If an STL operation is performed on the subtree rooted at node t_i as per Rule 5.1, then the resulting tree, $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$, is a partitioning BST.

Proof. To clarify the notation of the nodes involved, t_i is an internal node, whose left child, t_k, has a sibling which is the parent of t_j , and additionally, t_k and t_j are adjacent leaves.

We have to show that $\mathcal{T}'$, the tree obtained after invoking Rule 5.1, satisfies the properties of Definition 5.1. There are three issues which must be proven:



(a) The partitioning BST before applying the STL-1 operation.



(b) The resulting partitioning BST after the STL-1 operation.

Figure 5.4: An example showing an STL (Case STL-1) operation performed on a partitioning BST, $\mathcal{T}$, constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$. The resulting partitioning BST, $\mathcal{T}'$, is the one of (b). The dotted line above the topmost node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

(a) Each leaf node of $\mathcal{T}'$ is at position $2u-1$, $1 \leq u \leq m$.

(b) Each internal node of $\mathcal{T}'$ is at position $2u$, $1 \leq u \leq m$.

(c) The weight of each internal node, τ'_{2u}, in $\mathcal{T}'$, must be obtained as $\tau_{L_{2u}} + \tau_{R_{2u}}$, where L_{2u} and R_{2u} are the left and right children of t_{2u} respectively.

Without loss of generality, let t_i be t_{2u}, for some integer u, $1 \leq u \leq m$. This implies that $t_k = t_{2u-1}$ in $\mathcal{T}$. Besides $P_{P_j} = t_i$, which implies that $R_i = P_j$, and hence the next node to be enumerated after t_{2u} in $\mathcal{T}$ is $t_j = t_{2u+1}$. As a result, $P_j = t_{2u+2}$ is the next node to be enumerated after t_j in $\mathcal{T}$. In summary: $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, where $t_{2u-1} = t_k$, $t_{2u} = t_i$, $t_{2u+1} = t_j$, and $t_{2u+2} = P_j$ (see Figure 5.4).

(a) **Identity of Leaf nodes:** After performing the STL-1 operation on t_i, P_j will become the left child of t_i, which implies that P_j will continue to be an internal node, since t_k and t_j become its left and right children respectively. Since t_k and t_j are leaves, they will correspond to t'_{2u-1} and t'_{2u+1} in $\mathcal{T}'$, the modified tree, respectively. Additionally, all the other leaves, $\{t'_1, \ldots, t'_{2u-3}, t'_{2u+3}, \ldots, t'_{2m-1}\}$, remain unchanged in $\mathcal{T}'$, and hence (i) of Definition 5.1 will be satisfied.

(b) **Identity of Internal nodes:** Since P_j becomes the left child of t_i in $\mathcal{T}'$, P_j and t_i will be t'_{2u} and t'_{2u+2} respectively in $\mathcal{T}'$. Besides all the other internal nodes of $\mathcal{T}'$, $\{t'_2, \ldots, t'_{2u-2}, t'_{2u+4}, \ldots, t_{2m-2}\}$, remain unchanged, and hence (ii) of Definition 5.1 is satisfied.

(c) **Updating of τ:** On the other hand, $\tau_{P_j} = \tau'_{2u}$ is the only weight that is modified in $\mathcal{T}'$. Since t_k becomes the left child of P_j, τ_k is added to τ_{P_j} (this is done in (a) of STL-1). Additionally, after performing the STL, t_j remains as a child of P_j, and B_j is not a child of P_j anymore. Consequently, since τ_{B_j} is subtracted in (a) of STL-1, τ_{P_j} becomes the sum of $\tau_k = \tau_{2u-1}$ and $\tau_j = \tau_{2u+1}$, and the updated value of τ'_{P_j} is:

$$\tau'_{P_j} \leftarrow \tau_{P_j} + \tau_k - \tau_{B_j} .$$

The remaining weights of $\mathcal{T}'$, $\{\tau'_1, \ldots, \tau'_{2u-2}, \tau'_{2u+2}, \ldots, \tau'_{2m-2}\}$, are unchanged, and hence all the properties of Definition 5.1 are satisfied.

The result follows. □

From the proof of Lemma 5.2, we see that the weights of the internal nodes in the new tree, $\mathcal{T}'$, are consistently obtained as the sum of the weights of their two children. This is achieved in only *two* local operations, as opposed to re-calculating *all* the weights of the tree in a bottom-up fashion.

We now provide the mechanisms required to perform an STL operation when we are in the scenario of Case STL-2.

Rule 5.2 (STL-2). Consider a partitioning tree, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, and the notation of **Notation STL**. Suppose that we are in the scenario of Case STL-2. The STL operator performed on node t_i involves the following operations:

(a) $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) τ_j is subtracted from all the τ's in the path from P_{P_j} to R_i,

(c) k_i and k_{P_j} are swapped,

(d) B_j becomes the left child of P_{P_j},

(e) t_j becomes the right child of P_j,

(f) t_k becomes the left child of P_j, and

(g) P_j becomes the left child of t_i.

□

The following example will help to understand how the STL operation works in the scenario of Case STL-2.

Example 5.4. Let $\mathcal{S} = \{a, b, c, d, e\}$ be the source alphabet whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3]$. Suppose that the partitioning BST that we construct from $\mathcal{S}$ and $\mathcal{P}$ is the one depicted in Figure 5.5(a). The dashed lines indicate the changes needed for the STL operation.

The resulting partitioning BST, $\mathcal{T}'$, obtained after performing an STL operation (Case STL-2) on node t_i is depicted in Figure 5.5(b). Note that the resulting tree is a partitioning BST. The general case is stated and proved in Lemma 5.2 given below. □

Lemma 5.3 (STL-2 validity). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, as per **Notation STL**. The resulting tree, $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$, obtained after performing an STL-2 operation as per Rule 5.2 is a partitioning BST.

Proof. The proof of this lemma follows the steps of that of Lemma 5.2.

As in the proof of Lemma 5.2, we have to show that $\mathcal{T}'$, the tree obtained after invoking Rule 5.2, satisfies the properties of Definition 5.1. We again assume that: $t_{2u-1} = t_k$, $t_{2u} = t_i$, $t_{2u+1} = t_j$, and $t_{2u+2} = P_j$ (see Figure 5.5). We have to prove the following.

(a) **Identity of Leaf nodes:** This part of the proof is identical to (a) of Lemma 5.2.

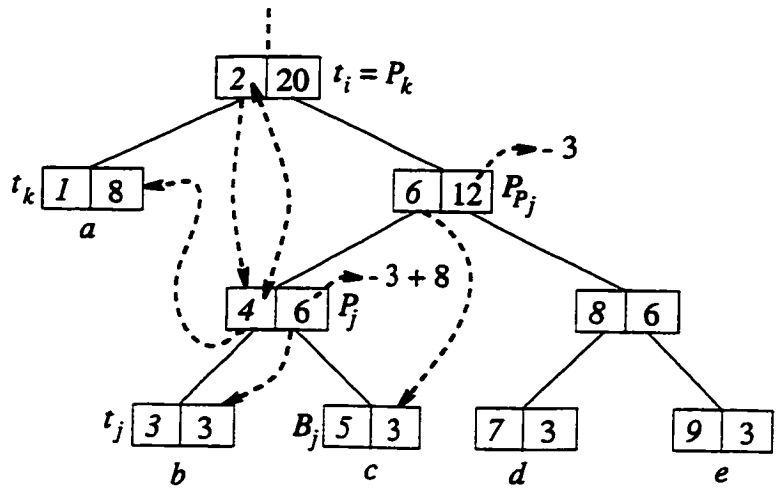
(b) **Identity of Internal nodes:** In this case, the proof follows the exact same steps of (b) of Lemma 5.2.

(c) **Updating of τ:** The updating of τ_{P_j} is achieved, as shown in (c) of Lemma 5.2. We prove the consistency of the operation.

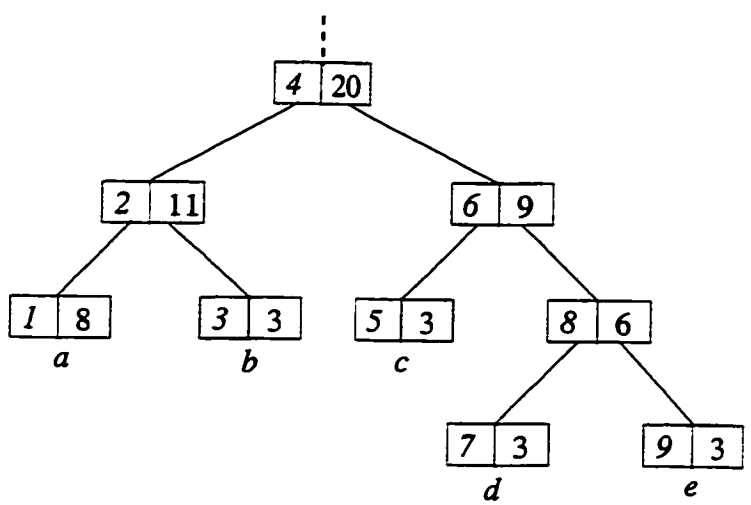
First of all, note that after invoking STL-2, P_j becomes the left child of t_i in $\mathcal{T}'$, and t_j becomes the right child of P_j. This implies that t_j is in the subtree rooted at L_i, and consequently, τ_j must be *subtracted* from the weights of *all* the nodes in the path from P_{P_j} to R_i. This is done in step (b) of Rule STL-2. Since t_j is not in the subtree rooted at R_i in $\mathcal{T}'$ (after the rotation), we see from (5.11) that, $\tau_{2u} = \sum_{v=1}^{s} \tau_{2v-1}$, and hence all the nodes in that path satisfy $\tau'_{2u} = \tau'_{L_{2u}} + \tau'_{R_{2u}}$.

Consequently, $\mathcal{T}'$ satisfies the properties of Definition 5.1, and the lemma follows.

□



(a) The partitioning BST before performing the STL-2 operation.



(b) The resulting partitioning BST after performing the STL-2 operation.

Figure 5.5: In (a), a partitioning BST is constructed from the source alphabet $\mathcal{S} = \{a, b, c, d, e\}$ whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3]$. In (b), the partitioning BST is obtained after performing an STL operation (Case STL-2) on node t_i. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

The corresponding rule for the scenario of Case STL-3 satisfying **Notation STL** is given below in Rule 5.3.

Rule 5.3 (STL-3). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, specified using the notation of **Notation STL**, and the scenario of Case STL-3. The STL operator performed on the subtree rooted at t_i consists of shifting t_j to the subtree rooted at L_i in such a way that:

(a) $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) τ_j is subtracted from all the τ's in the path from P_{P_j} to R_i,

(c) τ_j is added to all the τ's in the path from P_k to L_i,

(d) k_i and k_{P_j} are swapped,

(e) B_j becomes the left child of P_{P_j},

(f) t_j becomes the right child of P_j,

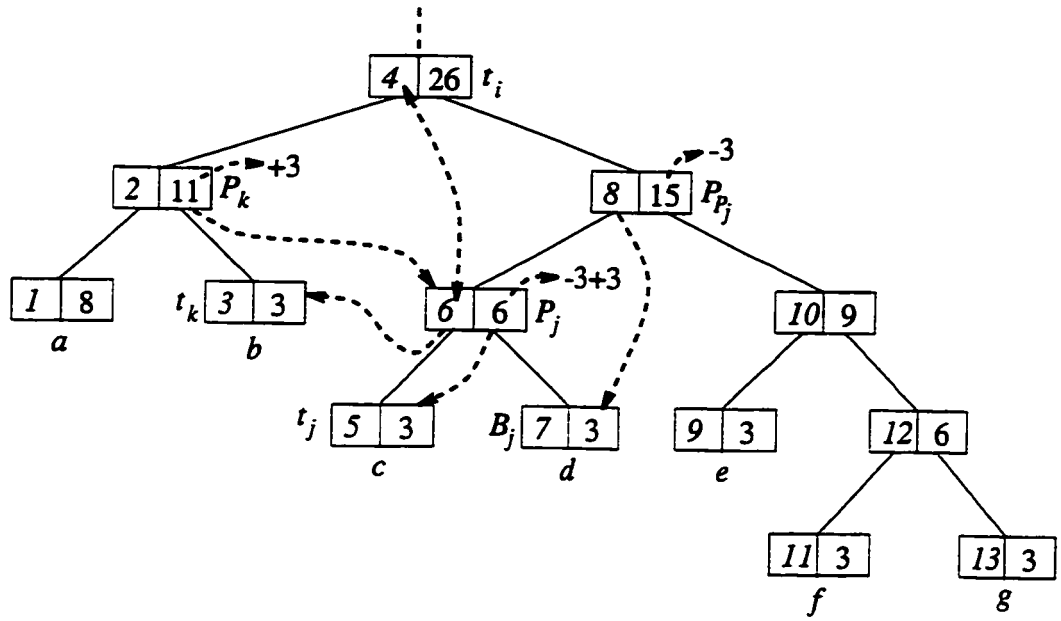
(g) t_k becomes the left child of P_j, and

(h) P_j becomes the right child of P_k.

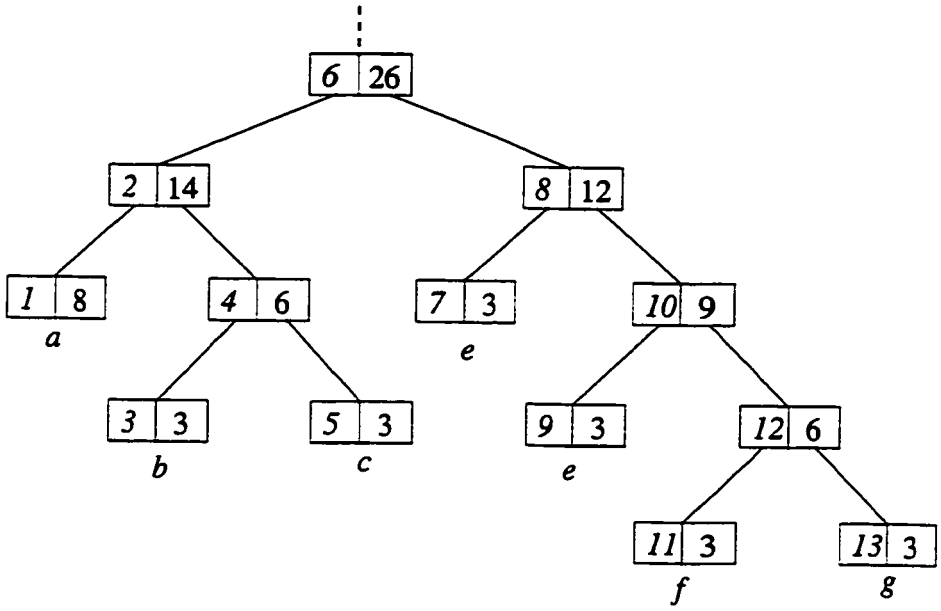
□

Observe that in the STL-3 operation, all the nodes in the entire path from P_k to the left child of t_i have to be updated by adding τ_j to the weight of those nodes. As in the other two cases, the weight of t_i is not changed. We now present an example that helps to understand how the STL-3 operator works.

Example 5.5. Let $\mathcal{S} = \{a, b, c, d, e, f, g\}$ be the source alphabet whose frequency counters are $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$. A partitioning BST, $\mathcal{T}$, constructed from $\mathcal{S}$ and $\mathcal{P}$ is the one depicted in Figure 5.6(a). After applying the STL operator to the subtree rooted at node t_i (in this case, the root node of $\mathcal{T}$), we obtain $\mathcal{T}'$, the tree depicted in Figure 5.6(b). Observe that $\mathcal{T}'$ is a partitioning BST. The general result is stated and proved in Lemma 5.4 given below. □



(a) The partitioning BST, $\mathcal{T}$, before performing the STL (Case STL-3) operation.



(b) The resulting tree, $\mathcal{T}'$, after the STL-3 operation is performed. $\mathcal{T}'$ is a partitioning BST.

Figure 5.6: A partitioning BST, $\mathcal{T}$, constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [8, 3, 3, 3, 3, 3, 3]$, and the corresponding partitioning BST, $\mathcal{T}'$, after performing an STL-3 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

Lemma 5.4 (STL-3 validity). Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a partitioning BST in which an STL-3 operation is performed as per Rule 5.3, resulting in a new tree, $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$. Then, $\mathcal{T}'$ is a partitioning BST.

Proof. The proof of this lemma follows the steps of that of Lemma 5.2.

As in the proof of Lemma 5.2, we have to show that $\mathcal{T}'$, the tree obtained after invoking Rule STL-3, satisfies the properties of Definition 5.1. We again assume the notation that: $t_{2u-1} = t_k$, $t_{2u} = t_i$, $t_{2u+1} = t_j$, and $t_{2u+2} = P_j$ (see Figure 5.6). We have to prove the following.

(a) **Identity of Leaf nodes:** This part of the proof is identical to (a) of Lemma 5.2.

(b) **Identity of Internal nodes:** In this case, the proof follows the exact same steps of (b) of Lemma 5.2.

(c) **Updating of τ:** The updating of τ_{P_j} and the weights of all the nodes in the path from P_{P_j} to R_i is achieved, as shown in (c) of Lemma 5.3. This is proved below.

In the case of STL-3, the parent of t_k, P_k, is not t_i, and hence, an additional operation is performed, which consists of *adding* τ_j to all the τ's in the path from P_k to L_i. Since t_j is incorporated into the subtree rooted at L_i in $\mathcal{T}'$, we see from (5.11), that $\tau_{2u} = \sum_{v=1}^{s} \tau_{2v-1}$. Thus, all the nodes in that path satisfy $\tau'_{2u} = \tau'_{L_{2u}} + \tau'_{R_{2u}}$. This implies that $\mathcal{T}'$ will satisfy all the properties of Definition 5.1.

The result follows. □

In order to facilitate the implementation of the STL operator, we present the corresponding algorithm that considers the three mutually exclusive cases discussed above. This procedure is depicted in Algorithm **STL_Operation**. When performing an assignment operation by means of the left arrow, "$\leftarrow$", the operand on the left is the new value of the pointer or weight, and the operand on the right is either the value of the weight, or the value of the actual pointer to a node. For example, $L_{P_{P_j}} \leftarrow B_j$ implies that the value of the pointer to the left child of P_{P_j} acquires the value "B_j"[6].

[6]In the actual implementation, "B_j" contains the memory address of the node B_j , or the position of B_j in a list if the tree is stored in an array.

Algorithm 14 STL_Operation

Input: A partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$.
The node on which the STL is performed, t_i.
The left-most leaf of the subtree rooted at R_i, t_j.
The right-most leaf of the subtree rooted at L_i, t_k.
Output: The modified partitioning BST, $\mathcal{T}$.
Assumptions: Those in **Notation STL**.
Method:

```
procedure STL(var T : partitioningBST; t_i, t_j, t_k : node);
  k_i ↔ k_{P_j}   // swap keys
  τ_{P_j} ← τ_{P_j} + τ_k − τ_{B_j}
  for l ← P_{P_j} to R_i step l ← P_l do
    τ_l ← τ_l − τ_j
  endfor
  for l ← P_k to L_i step l ← P_l do
    τ_l ← τ_l + τ_j
  endfor
  if t_i = P_k then   // Move P_j to the subtree on the left
    L_i ← P_j   // STL-1 and STL-2
  else
    R_{P_k} ← P_j   // STL-3
  endif
  if t_i = P_{P_j} then   // B_j remains in the subtree on the right
    R_i ← B_j   // STL-1
  else
    L_{P_{P_j}} ← B_j   // STL-2 and STL-3
  endif
  R_{P_j} ← t_j   // t_j becomes the right child of its parent
  L_{P_j} ← t_k; P_k ← P_j   // P_j becomes the parent of t_k
  P_{B_j} ← P_{P_j}   // Update the parent of B_j
  P_{P_j} ← P_k   // The new parent of P_j is now the old parent of t_k
endprocedure
```

end Algorithm STL_Operation

5.3.3.2 The Shift-To-Right Operator

The STL operator and the STR operator use similar principles for shifting nodes. Broadly speaking, the STR operator applied to an internal node, t_i, consists of removing the right-most leaf of the subtree rooted at the left child of t_i, and inserting it as the left-most leaf into the subtree rooted at the right child of t_i.

Consider a list, $\mathcal{P} = [p_1, \ldots, p_m]$, which has been partitioned into two sublists, $\mathcal{P}_0 = [p_1, \ldots, p_k]$ and $\mathcal{P}_1 = [p_{k+1}, \ldots, p_m]$. The STR operator applied to an internal node of a partitioning BST, is equivalent to removing the last element of $\mathcal{P}_0$, and inserting it as the first element of $\mathcal{P}_1$. After applying the STR operation, the resulting sublists are $\mathcal{P}_0' = [p_1, \ldots, p_{k-1}]$ and $\mathcal{P}_1' = [p_k, p_{k+1}, \ldots, p_m]$.

We now introduce the notation for the STR operator.

Notation STR: Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. Let

- t_i be an internal node of $\mathcal{T}$,
- L_i be also an internal node of $\mathcal{T}$,
- t_j be the right-most leaf of the subtree rooted at L_i, and
- t_k be the left-most leaf of the subtree rooted at R_i.

As in the STL operator, we identify three mutually exclusive cases in which the STR operator can be applied.

STR-1 : $P_{P_j} = t_i$ and R_i is a leaf.

STR-2 : $P_{P_j} \neq t_i$ and R_i is a leaf.

STR-3 : R_i is *not* a leaf.

We again state the rules for performing the STR operation for the three cases listed above. The STR operator performed in the scenario of Case STR-1 is formalized below.

Rule 5.4 (STR-1). Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a partitioning BST specified by the notation of **Notation STR**, and the scenario of Case STR-1. The STR operator applied to the subtree rooted at node t_i involves the following operations.

(a) $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) k_i and k_{P_j} are swapped,

(c) B_j becomes the left child of t_i,

(d) P_j becomes the right child of t_i,

(e) t_k becomes the right child of P_j, and

(f) t_j becomes the left child of P_j.

□

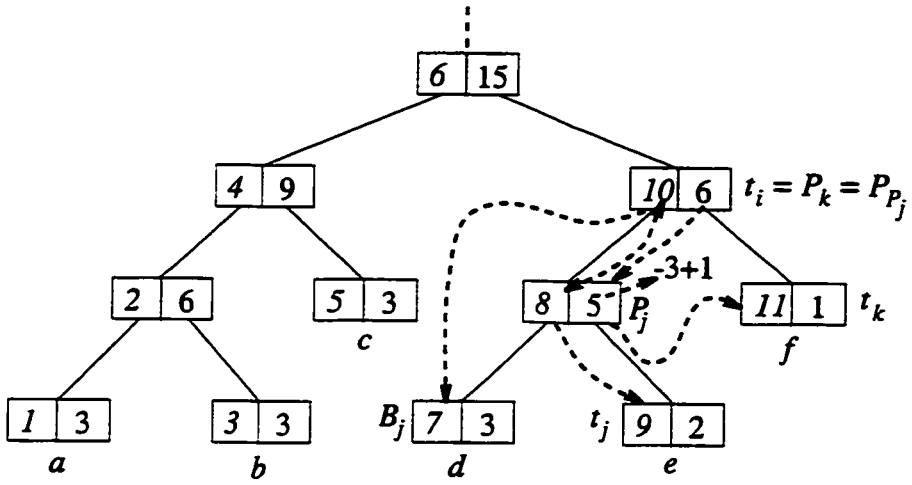
Remark 5.4. The node t_i can be any internal node of $\mathcal{T}$ (not necessarily the root) that is specified by **Notation STR**. The next example includes an STR operation performed on an internal, non-root node, t_i, in the scenario of Case STR-1.

Example 5.6. Let $\mathcal{S} = \{a, b, c, d, e, f\}$ be the source alphabet, and $\mathcal{P} = [3, 3, 3, 3, 2, 1]$ be the frequency counters. Suppose that $\mathcal{T}$ is the partitioning BST, constructed from $\mathcal{S}$ and $\mathcal{P}$, depicted in Figure 5.7(a). If we perform an STR-1 operation on the subtree rooted at node t_i, the resulting partitioning BST, $\mathcal{T}'$, is the one depicted in Figure 5.7(b). Observe that t_i is an internal, non-root node. After performing the STR operation, the resulting tree, $\mathcal{T}'$, is a partitioning BST. This is a general result, which is stated and proved in Lemma 5.5 given below. □

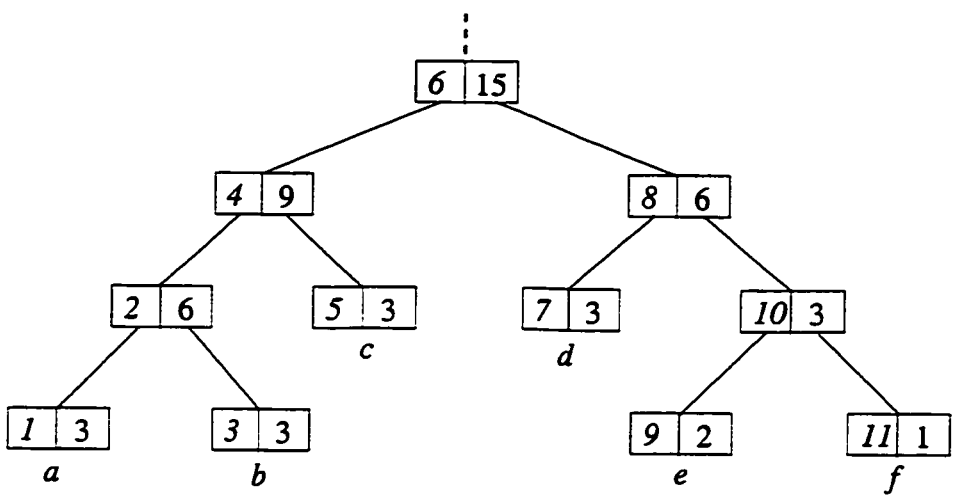
Lemma 5.5 (STR-1 validity). Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a partitioning BST. If $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$ is the resulting tree obtained after performing an STR-1 operation on t_i, then $\mathcal{T}'$ is a partitioning BST.

Proof. [7] First of all, we clarify the notation for the nodes involved in this proof, t_i is an internal node, whose right child, t_k, has a sibling which is the parent of t_j, and additionally, t_k and t_j are adjacent leaves.

[7] The proof of this lemma is similar to the proof of Lema 5.2 (STL-1 Validity). In this lemma, t_j and t_k interchange roles in $\mathcal{T}$, being t_{2u-1} and t_{2u+1} respectively.



(a) The partitioning BST before performing the STR-1 operation on node t_i. The dashed lines are the new pointers in the resulting tree.



(b) The resulting partitioning BST after performing the STR-1 operation.

Figure 5.7: A partitioning BST constructed from $\mathcal{S} = \{a, b, c, d, e, f\}$ and $\mathcal{P} = [3, 3, 3, 3, 2, 1]$. In (a), the partitioning BST before performing the STR-1 operation, and in (b), the resulting partitioning BST. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

In order to ensure the validity of Rule STR-1, we must show that the resulting tree, $\mathcal{T}'$, satisfies the properties of Definition 5.1. We achieve this by proving the following three statements:

(a) Each leaf node of $\mathcal{T}'$ is at position $2u-1$, $1 \leq u \leq m$.

(b) Each internal node of $\mathcal{T}'$ is at position $2u$, $1 \leq u \leq m$.

(c) The weight of each internal node, τ'_{2u}, in $\mathcal{T}'$, must be obtained as $\tau_{L_{2u}} + \tau_{R_{2u}}$, where L_{2u} and R_{2u} are the left and right children of t_{2u} respectively.

Without loss of generality, we let t_i be t_{2u}, for some integer u, where $1 \leq u \leq m$. As a consequence of this, the next node to be enumerated after t_{2u} in $\mathcal{T}$ will be $t_k = t_{2u+1}$. Besides $P_{P_j} = t_i$, which implies that $L_i = P_j$, and hence t_j (a leaf) will be t_{2u-1} in $\mathcal{T}$. As a result, P_j will be t_{2u-2} in $\mathcal{T}$. Thus, we have: $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, where $t_{2u-2} = P_j$, $t_{2u-1} = t_j$, $t_{2u} = t_i$, and $t_{2u+1} = t_k$ (see Figure 5.7).

(a) **Identity of Leaf nodes:** After performing the STR-1 operation on t_i, P_j will become the right child of t_i. This implies that P_j will remain as an internal node in $\mathcal{T}'$, because t_j and t_k become its left and right children respectively. Since t_j and t_k are both leaves, they will correspond to t'_{2u-1} and t'_{2u+1} in $\mathcal{T}'$, the resulting tree after performing the STR-1 operation, respectively. In addition, all the other leaves of $\mathcal{T}'$, $\{t'_1, \ldots, t'_{2u-3}, t'_{2u+3}, \ldots, t'_{2m-1}\}$, will remain unchanged. Therefore, (i) of Definition 5.1 is satisfied.

(b) **Identity of Internal nodes:** From STR-1, we see that P_j becomes the right child of t_i in $\mathcal{T}'$. Consequently, t_i and P_j will be t'_{2u-2} and t'_{2u} respectively in $\mathcal{T}'$. Additionally, all the other internal nodes, $\{t'_2, \ldots, t'_{2u-4}, t'_{2u+2}, \ldots, t_{2m-2}\}$, will remain unchanged in $\mathcal{T}'$, and hence (ii) of Definition 5.1 will be satisfied.

(c) **Updating of τ:** As a result of the STR-1 operation, $\tau_{P_j} = \tau'_{2u}$ is the only weight that is modified in $\mathcal{T}'$. Since t_k becomes the right child of P_j, τ_k is added to τ_{P_j} (this is achieved in Step (a) of Rule STR-1). In addition, after performing the STR-1 operation, t_j will remain as a child of P_j, and B_j will not be child of

P_j anymore. Therefore, since τ_{B_j} is subtracted in Step (a) of Rule STR-1, τ_{P_j} becomes the sum of $\tau_j = \tau_{2u-1}$ and $\tau_k = \tau_{2u+1}$, and hence the updated value of τ'_{P_j} is:

$$\tau'_{P_j} \leftarrow \tau_{P_j} + \tau_k - \tau_{B_j} .$$

Since the weights of *all* the other nodes in $\mathcal{T}'$, $\{\tau'_1, \ldots, \tau'_{2u-3}, \tau'_{2u+2}, \ldots, \tau'_{2m-2}\}$ remain unchanged, *all* the properties of Definition 5.1 will be satisfied.

The result follows. □

The formal definition of the operations required to perform an STR operation for the scenario of the second case, STR-2, is formalized below.

Rule 5.5 (STR-2). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, specified as per **Notation STR**, and the scenario of Case STR-2. The STR operator applied on t_i involves the following operations:

(a) $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) τ_j is subtracted from all the τ's from P_{P_j} to L_i,

(c) k_{P_j} and k_i are swapped,

(d) B_j becomes the right child of P_{P_j},

(e) P_j becomes the right child of t_i,

(f) t_j becomes the left child of P_j, and

(g) t_k becomes the right child of P_j.

□

We now provide an example that includes an STR operation performed on a partitioning BST and the scenario of Case STR-2.

Example 5.7. Consider the source alphabet $\mathcal{S} = \{a, b, c, d\}$ whose frequency counters are $\mathcal{P} = \{2, 2, 2, 1\}$. Let $\mathcal{T}$ be the partitioning BST depicted in Figure 5.8(a). The dashed arcs are the changes produced by performing the STR-2 operation. The resulting tree, $\mathcal{T}'$, is that of Figure 5.8(b). $\mathcal{T}'$ is a partitioning BST. The general result is stated and proved in Lemma 5.6 given below. □

Lemma 5.6 (STR-2 validity). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, specified as per **Notation STR**. An STR-2 operation on t_i produces a partitioning BST, $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$.

Proof. This proof is similar to the proof of Lemma 5.5.

We are to show that the tree obtained after invoking Rule STR-2, $\mathcal{T}'$, satisfies the properties of Definition 5.1. In this proof, we again use the notation that: $t_{2u-2} = P_j$, $t_{2u-1} = t_j$, $t_{2u} = t_i$, and $t_{2u+1} = t_k$ (see Figure 5.8). We are to prove the following three statements.

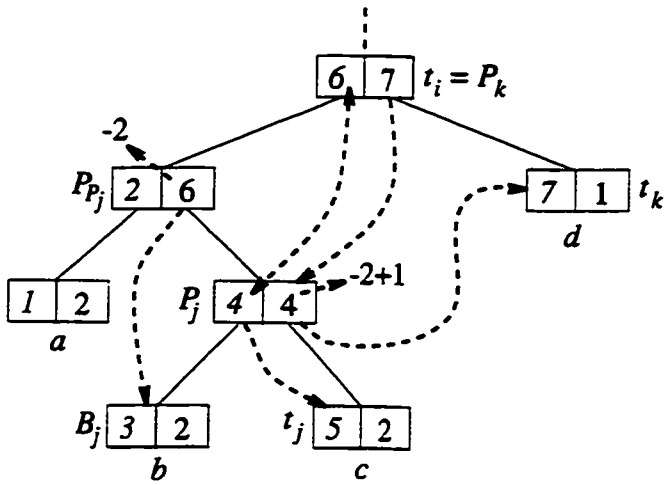
(a) **Identity of Leaf nodes:** This part of the proof is identical to (a) of Lemma 5.5.

(b) **Identity of Internal nodes:** In this case, the proof follows the exact same steps of (b) of Lemma 5.5.

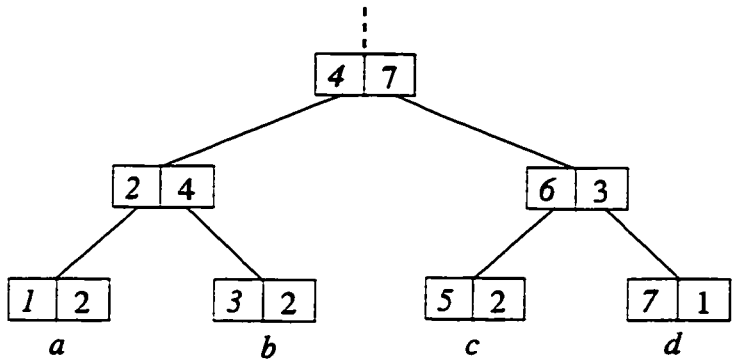
(c) **Updating of τ:** The updating of τ_{P_j} is achieved as stated in the STR-2 rule. We now prove the consistency of the operation.

We first see that after invoking STR-2, P_j becomes the right child of t_i in $\mathcal{T}'$. Also, t_j becomes the left child of P_j in $\mathcal{T}'$. Consequently, t_j is in the subtree rooted at R_i, and hence τ_j must be *subtracted* from the weights of *all* the nodes in the path from P_{P_j} to L_i. This is consistently achieved in Step (b) of Rule STR-2. Since t_j is not in the subtree rooted at L_i in $\mathcal{T}'$ (after the rotation), from (5.11), we see that $\tau_{2u} = \sum_{v=1}^{s} \tau_{2v-1}$. Therefore, all the nodes in that path satisfy $\tau'_{2u} = \tau'_{L_{2u}} + \tau'_{R_{2u}}$.

$\mathcal{T}'$ thus satisfies all the properties of Definition 5.1, and the lemma follows. □



(a) The partitioning BST before performing the STR-2 operation on t_i.



(b) The resulting tree after performing the STR-2 operation.

Figure 5.8: A partitioning BST constructed from the source alphabet $\mathcal{S} = \{a, b, c, d\}$ and $\mathcal{P} = [2, 2, 2, 1]$, and the corresponding tree after performing an STR-2 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

The last case that we consider for performing STR operations on partitioning BSTs is defined below.

Rule 5.6 (STR-3). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, with the notation of **Notation STR**, and the scenario of Case STR-3. Applying an STR operator on t_i consists of the following operations:

(a) $\tau_k - \tau_{B_j}$ is added to τ_{P_j},

(b) τ_j is subtracted from all the τ's from P_{P_j} to L_i,

(c) τ_j is added to all the τ's from P_k to R_i,

(d) k_i and k_{P_j} are swapped,

(e) B_j becomes the right child of P_{P_j},

(f) t_j becomes the left child of P_j,

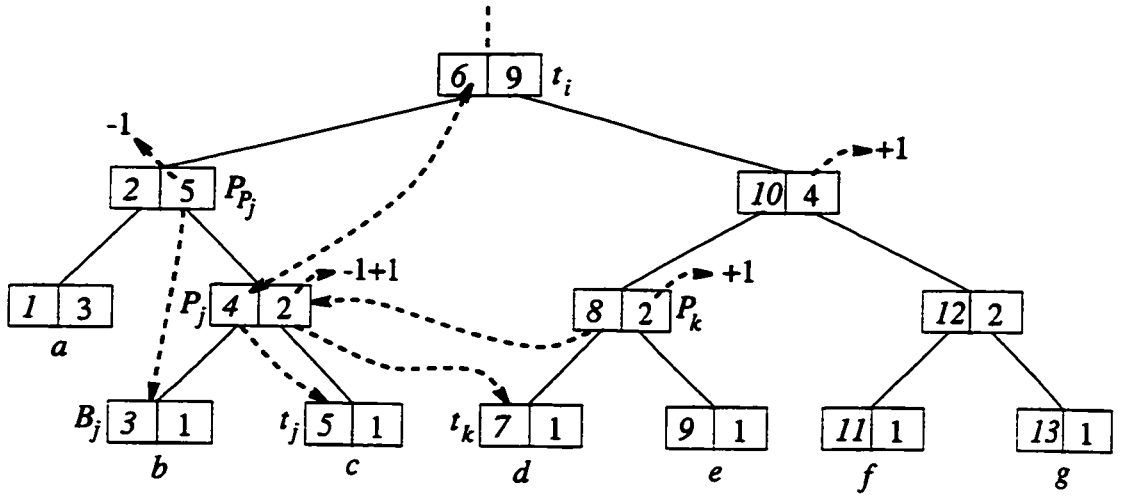
(g) t_k becomes the right child of P_j, and

(h) P_j becomes the left child of P_k.

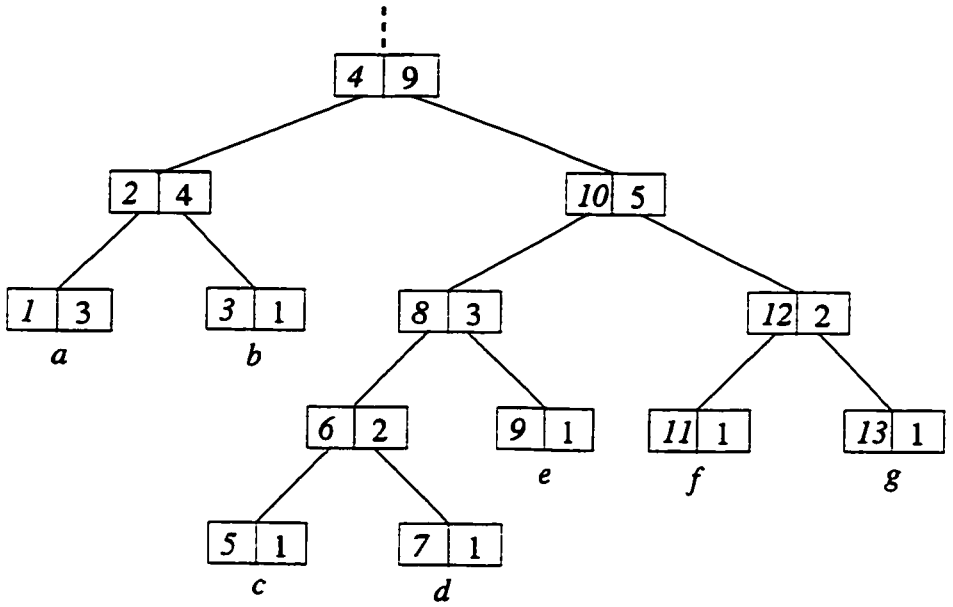
□

The next example depicts how the STR operator works in the scenario of Case STR-3.

Example 5.8. Let $\mathcal{S} = \{a, b, c, d, e, f, g\}$ be the source alphabet whose frequency counters are $\mathcal{P} = [3, 1, 1, 1, 1, 1, 1]$. Let $\mathcal{T}$ be the partitioning BST shown in Figure 5.9(a). Suppose that we perform an STR-3 operation on node t_i. The changes to $\mathcal{T}$ are marked with dashed arcs, and the resulting tree is the one shown in Figure 5.9(b). The resulting tree is a partitioning BST. Again, this is a general result, which is stated and proved in Lemma 5.7 given below. □



(a) The partitioning BST before perfoming the STR-3 operation. The dashed arcs indicate the changes to be done by the STR operator.



(b) The resulting partitoning BST after the STR-3 operation.

Figure 5.9: Two partitioning BSTs. The tree on the top was constructed from $\mathcal{S} = \{a, b, c, d, e, f, g\}$ and $\mathcal{P} = [3, 1, 1, 1, 1, 1, 1]$, and includes the changes to be done when applying an STR-3 operation on t_i. The tree on the bottom is the resulting partitioning BST after the STR-3 operation. The dotted line above the top-most node indicates that this node could be a left child or a right child of its parent, or could be the root node itself.

Lemma 5.7 (STR-3 validity). Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, specified as per **Notation STR**, and the scenario of Case STR-3. The tree $\mathcal{T}' = \{t'_1, \ldots, t'_{2m-1}\}$, which results from applying an STR operator on t_i, is a partitioning BST.

Proof. The proof of this lemma follows the steps of the proof of Lemma 5.5.

As in the proof of Lemma 5.5, we are to show that $\mathcal{T}'$, the tree obtained after invoking Rule STR-3, satisfies the properties of Definition 5.1. We again use the following notation: $t_{2u-2} = P_j$, $t_{2u-1} = t_j$, $t_{2u} = t_i$, and $t_{2u+1} = t_k$ (see Figure 5.9). We prove the following three statements.

(a) **Identity of Leaf nodes:** This part of the proof follows the exact same steps of (a) of Lemma 5.5.

(b) **Identity of Internal nodes:** In this case, the proof is identical to part (b) of the proof of Lemma 5.5.

(c) **Updating of τ:** The updating of τ_{P_j} and the weights of all the nodes in the path from P_{P_j} to R_i is achieved, as shown in part (c) of the proof of Lemma 5.6. We prove the correctness of this operation.

When invoking an STR-3 operation, the parent of t_k, P_k, is different from t_i. This implies that an additional operation is performed, which consists of *adding* τ_j to all the τ's in the path from P_k to R_i. Since t_j is incorporated into the subtree rooted at R_i in $\mathcal{T}'$, from (5.11), it follows that $\tau_{2u} = \sum_{v=1}^{s} \tau_{2v-1}$. Consequently, all the nodes in that path will satisfy $\tau'_{2u} = \tau'_{L_{2u}} + \tau'_{R_{2u}}$. Therefore, $\mathcal{T}'$ will satisfy all the properties of Definition 5.1.

The lemma thus follows. □

The implementation of the three cases in which the STR operator can be applied is included in Algorithm **STR_Operation** given below. This algorithm contains the procedure STR(...), which takes a partitioning BST, $\mathcal{T}$, and a node on which the STR operation is performed, generating a new partitioning BST. We refer to the new partitioning BST as $\mathcal{T}$, and we use the modifier **var** so as to return the new tree.

Algorithm 15 STR_Operation

Input: A partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$.
The node on which the STR is performed, t_i.
The right-most leaf of the subtree rooted at L_i, t_j.
The left-most leaf of the subtree rooted at R_i, t_k.
Output: The modified partitioning BST, $\mathcal{T}$.
Assumptions: Those in **Notation STR**.
Method:

```
procedure STL(var T : partitioningBST; t_i, t_j, t_k : node);
  k_i ↔ k_{P_j}   // swap keys
  τ_{P_j} ← τ_{P_j} + τ_k − τ_{B_j}
  for l ← P_{P_j} to L_i step l ← P_l do
    τ_l ← τ_l − τ_j
  endfor
  for l ← P_k to R_i step l ← P_l do
    τ_l ← τ_l + τ_j
  endfor
  if t_i = P_k then   // Move P_j to the subtree on the right
    R_i ← P_j   // STR-1 and STR-2
  else
    L_{P_k} ← P_j   // STR-3
  endif
  if t_i = P_{P_j} then   // B_j remains in the subtree on the right
    L_i ← B_j   // STR-1
  else
    R_{P_{P_j}} ← B_j   // STR-2 and STR-3
  endif
  L_{P_j} ← t_j   // t_j becomes the left child of its parent
  R_{P_j} ← t_k; P_k ← P_j   // P_j becomes the parent of t_k
  P_{B_j} ← P_{P_j}   // Update the parent of B_j
  P_{P_j} ← P_k   // The new parent of P_j is now the old parent of t_k
endprocedure
```

end Algorithm STR_Operation

5.3.4 The Conditional Shifting Heuristic

Our task is to maintain a Fano BST at all times - at the encoding and decoding stages. In the encoding algorithm, at time 'k', the symbol $x[k]$ is encoded using a Fano BST, $\mathcal{T}(k)$, which is identical to the one used by the decoding algorithm to retrieve $x[k]$. $\mathcal{T}(k)$ must be updated in such a way that at time '$k+1$', both algorithms maintain the same tree $\mathcal{T}(k+1)$. In this section, we present the formal procedure that takes a Fano BST and the node corresponding to the current symbol occurring in the input sequence, and generates a new Fano BST. This procedure is called updateFanoBST(...), which will be invoked by both Algorithms **Fano_BST_Encoding** and **Fano_BST_Decoding** presented subsequently.

The aim of our approach is to maintain, at each time instant, a Fano BST, i.e. the tree obtained after the updating procedure, $\mathcal{T}(k+1)$, must satisfy the conditions stated in Definition 5.2. We know that the Fano BST is a particular case of the partitioning BST, and, as we have shown in Section 5.3.3, after performing a shifting operation (either an STL or an STR) on a partitioning BST, the resulting tree is also a partitioning BST. Therefore, it is up to the updating procedure to ensure that the resulting tree satisfies the conditions stated in Definition 5.2.

First of all, we introduce a conditional shifting heuristic used to transform a partitioning BST into a Fano BST. The conditional shifting heuristic is based on the principles of the Fano coding – the nearly-equal-probability property defined in Chapter 4. This heuristic, used in conjunction with the STL and the STR operators defined in this chapter, are used to transform a partitioning BST into a Fano BST. For every internal node, and based on the conditional shifting heuristic, the shifting operations are performed as per the following rule.

Rule 5.7. Consider a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. Let t_i be an internal node of $\mathcal{T}$, t_j be the left-most leaf of the subtree rooted at R_i, and t_k be the right-most leaf of the subtree rooted at L_i.

(*i*) If

$$\theta_1 = \tau_{R_i} - \tau_{L_i} - \tau_j > 0\,, \tag{5.17}$$

then perform an STL operation on t_i.

(*ii*) If

$$\theta_2 = \tau_{L_i} - \tau_{R_i} - \tau_k > 0\,, \tag{5.18}$$

then perform an STR operation on t_i.

□

We now introduce a definition that is important in the formalization of the algorithms that implements the updating procedure. We let a *two-leaf internal node* be an internal node whose left and right children are leaves.

Definition 5.3. Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a partitioning BST. An internal node, t_i, is a *two-leaf internal node* if and only if L_i and R_i are both leaves.

The procedure for updating[8] the Fano BST, procedure updateFanoBST(...), is formalized in Algorithm **Fano_BST_Updating**. This procedure receives as parameters a partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, the root node, *root*, and the node associated with the symbol being encoded, t_n. The weight of t_n at time 'k', $\tau_n(k)$, is updated by adding unity to it. In order to maintain the source symbols of $\mathcal{T}$ in a non-increasing order of probability from left to right, t_n is swapped with the left-most leaf (if there is any) whose weight is less than $\tau_n(k)$. Let $\hat{\mathcal{P}}(k) = \{\hat{p}_1(k), \ldots, \hat{p}_m(k)\}$ be the estimated probabilities of $\mathcal{S}$, and $\eta(k) = \{\eta_1(k), \ldots, \eta_m(k)\}$ be the frequency counters of $\mathcal{S}$. In order to avoid duplicating operations, $\hat{\mathcal{P}}(k)$ is not updated. In fact, incrementing the weight of t_n by unity is equivalent to incrementing $\eta_i(k)$ by unity, and consequently updating $\hat{p}_i(k) = \frac{\eta_i(k)}{\eta+k} = \frac{\tau_n(k)}{\eta+k}$, where s_i is the symbol associated with t_n, and $\eta = \sum_{i=1}^{m} \eta_i(1)$. We show later in this chapter that using this updating procedure, the adaptive Fano BST asymptotically converges to the static Fano tree.

[8]We are grateful to Dr. Kaizhong Zhang for his help in correcting procedure checkShift(...). The corrections involved introducing two "while" statements in place of the corresponding previous "if" statements.

Algorithm 16 Fano_BST_Updating

Input: A partitioning BST, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. The root of $\mathcal{T}$, *root*.
The leaf node associated with $x[k]$ whose weight has to be updated, t_n.
Output: A Fano BST, $\mathcal{T}$.
Method:

```
procedure updateFanoBST(T : partitioningBST; root, t_n : node);
  τ_n ← τ_n + 1   // Increment the weight of t_n
  Swap t_n with the left-most leaf, t_v, where τ_n > τ_v
  t_i ← t_n
  while t_i ≠ root do   // Walk up towards the root
    t_i ← P_i
    τ_i ← τ_i + 1   // Update the internal node weights
  endwhile
  while k_i ≠ k_n do   // Walk downwards t_n
    checkShift(T, t_i)
    if k_n < k_i then
      t_i ← L_i   // Go to the left child
    else
      t_i ← R_i   // Go to the right child
    endif
  endwhile
endprocedure
procedure checkShift(T : partitioningBST; t_i : node);
  if t_i ≠ 'two-leaf internal node' then return endif
  t_j ← left-most leaf of the subtree rooted at R_i
  t_k ← right-most leaf of the subtree rooted at L_i
  while τ_{R_i} − τ_{L_i} − τ_j > 0 do   // θ_1 > 0
    STL(T, t_i, t_j, t_k)
    checkShift(T, L_i)
    checkShift(T, R_i)
  endwhile
  while τ_{L_i} − τ_{R_i} − τ_k > 0 do   // θ_2 > 0
    STR(T, t_i, t_k, t_j)   // In the STR, t_j and t_k interchange roles
    checkShift(T, L_i)
    checkShift(T, R_i)
  endwhile
endprocedure
```

end Algorithm Fano_BST_Updating

The weights of all the internal nodes in the path from P_n to the root are updated. After this, the path from the root downwards t_n is inspected to see if there is a non-two-leaf internal node that does not satisfy the conditions stated in Definition 5.2. This is achieved by invoking the procedure checkShift(...) explained below. The path from the root downwards t_n is traced by using the key of each internal node, in such a way that the key of t_n, k_n, is searched as in a binary search tree.

The procedure checkShift(...) of Algorithm **Fano_BST_Updating** is responsible for checking that all the non-two-leaf internal nodes satisfy Definition 5.2. This procedure receives as parameters the partitioning BST, $\mathcal{T}$, and the non-two-leaf internal node to be inspected, t_i. By following Rule 5.7, if (5.17) is true (i.e. $\theta_1 > 0$), an STL operator is performed on t_i, and the procedure checkShift(...) is recursively invoked for the left and right children of t_i. If $\theta_1 \leq 0$, (5.18) is evaluated, and then if $\theta_2 > 0$, an STR operation is performed on t_i, and, as in the case of the STL operation, the procedure checkShift(...) is recursively invoked for the left and right children of t_i.

After the procedure checkShift(...) is executed on all the non-two-leaf internal nodes of the path from the root to the leaf associated with $x[k]$, t_n, the partitioning BST is transformed into a Fano BST. This property as well as the validity of the encoding and decoding algorithms are analyzed later in this chapter.

5.3.5 The Encoding Algorithm

In this section, we present the encoding algorithm, which takes the source sequence and generates the encoded sequence by adaptively maintaining a Fano BST. This procedure is given in Algorithm **Fano_BST_Encoding** given below.

All the elements of the list of frequency counters, $\mathcal{P}$, are initialized to unity. Other arbitrary conventions can also be applied, e.g. $p_i \leftarrow 0.2$ (or for that matter, any real number). However, initializing all the p_i's to zero causes some problems when updating the tree, or when calculating probabilities from the frequency counters.

The initial Fano BST is created by invoking procedure FanoBST(...) of Algorithm **Fano_BST_Construction** presented in Section 5.3.2, which uses $\mathcal{S}$ and $\mathcal{P}$. Note that we send $\eta(1)$ as a parameter instead of $\mathcal{P}$. In fact, the tree construction procedure

Algorithm 17 Fano_BST_Encoding

Input: The source alphabet, $\mathcal{S}$. A source sequence, $\mathcal{X} = x[1] \ldots x[M]$.
Output: The encoded sequence $\mathcal{Y} = y[1] \ldots y[R]$.
Assumptions: The Fano BST is constructed by invoking FanoBST(...), and maintained correctly by invoking updateFanoBST(...).
Method:

```
η_i(1) ← 1 for i = 1, ... , m
 Create node root
 η(1) ← Σ_{i=1}^{m} η_i(1); τ_root ← Σ_{i=1}^{m} η_i(1)
 FanoBST(S, η(1), 1, m, root, τ_root)
j ← 1
for k ← 1 to M do
   i ← position of x[k] in T from left to right (counting the leaves only)
   t_n ← root   // For each symbol, start again from the root
   while t_n ≠ "leaf" do
     if 2i − 1 < k_n then    // Perform a binary search
       y[j] ←'0'; t_n ← L_n  // Go to left child
     else
       y[j] ←'1'; t_n ← R_n  // Go to right child
     endif
     j ← j + 1
   endwhile
   updateFanoBST(T, root, t_n)
 endfor
end Algorithm Fano_BST_Encoding
```

works either with frequency counters or with probabilities. The Fano BST is accessed through its root node whose pointer is stored in the variable *root*.

Let $\mathcal{T}(k)$ be the Fano BST at time 'k', which, in the algorithm, is represented by *root*. The encoding process proceeds, as usual, by scanning all the symbols of $\mathcal{X}$ from left to right. At time 'k', i is obtained as the index of $x[k]$ in $\mathcal{S}$. By taking advantage of Property (i) of a Fano BST, $2i-1$ is searched in $\mathcal{T}(k)$, as if it were searched in a binary search tree : Going to the left child when $2i-1 < k_n$, or to the right child otherwise. The value $2i-1$ is always found, since as mentioned earlier, we assume that all the keys are already in the BST. Besides when going to the left child, '0' is sent to the output, and when going to the right child, '1' is sent to the output. Any of the 2^{m-1} labeling strategies other than the one used here can also be used.

Once the kth symbol from the input is encoded, $\mathcal{T}(k)$ must be updated. The updating procedure, updateFanoBST(...), involves two phases: increment the frequency counter of s_i, $\eta_i(k)$, and make the necessary changes in such a way that $\mathcal{T}(k+1)$ is a Fano BST. This procedure was discussed in Section 5.3.4.

5.3.6 The Decoding Algorithm

We now present the decoding algorithm that implements the adaptive version of the Fano decoding using Fano BSTs. The corresponding procedure is given in Algorithm **Fano_BST_Decoding**.

As in the encoding algorithm, the counters at time '1', $\eta(1) = [\eta_1(1), \ldots, \eta_m(1)]$, are initialized to unity. The initial Fano BST is created by invoking the procedure FanoBST(...) of Algorithm **Fano_BST_Construction** introduced in Section 5.3.2.

Let $\mathcal{T}(k)$ be the Fano BST used to decode $x[k]$. In the algorithm, this tree is represented by the variable *root*. The decoding procedure proceeds by scanning the encoded sequence, $\mathcal{Y}$, from left to right. An auxiliary pointer, n, is maintained, which stores the pointer to the current node being inspected at time 'j'. Each time a bit from the input, $y[j]$, is received, the pointer t_n is moved to the left child of t_n, L_n, if $y[j]$ is a '0', or to the right child of t_n, R_n, if $y[j]$ is a '1'. When a leaf node is reached, the symbol associated with it, $s_{\frac{n+1}{2}}$, is recovered and sent to the output. At this point the

Algorithm 18 Fano_BST_Decoding

Input: The source alphabet, $\mathcal{S}$. The encoded sequence $\mathcal{Y} = y[1] \ldots y[R]$.
Output: The original source sequence, $\mathcal{X} = x[1] \ldots x[M]$.
Assumptions: The Fano BST is constructed by invoking FanoBST(...), and maintained correctly by invoking updateFanoBST(...).
Method:

```
η_i(1) ← 1 for i = 1, ... , m
 Create node root
 η(1) ← Σ_{i=1}^{m} η_i(1); τ_root ← Σ_{i=1}^{m} η_i(1)
 FanoBST(S, η(1), 1, m, root, τ_root)
k ← 1
t_n ← root
for j ← 1 to R do
  if y[j] ='0' then   // Equivalent to the binary search done in the encoder
    t_n ← L_n  // Go to left child
  else
    t_n ← R_n  // Go to right child
  endif
  if t_n = "leaf" then
    x[k] ← s_{(n+1)/2}  // The symbol associated with t_n is retrieved
    k ← k + 1
    updateFanoBST(T, root, t_n)
  endif
endfor
end Algorithm Fano_BST_Decoding
```

tree is immediately updated by invoking the procedure updateFanoBST(...), discussed earlier, so that a Fano BST is maintained, which is the one used to decode the next source symbol, $x[k+1]$.

5.3.7 Properties of the Fano BST Coding

In this section, we study various properties of the Fano BST coding. First of all, we show that the tree obtained after performing the procedure updateFanoBST(...) is a Fano BST. We then analyze the correctness of Algorithms **Fano_BST_Encoding** and **Fano_BST_Decoding**. We show that if an input sequence is encoded using the former, decoding the resulting encoded sequence using the latter results in the original input sequence. The third property that we study is related to the asymptotic number of changes required to maintain a Fano BST for stationary distributions.

In Section 5.3.3, we have shown that after an STL or an STR operator is applied to an internal node of a partitioning BST, the resulting tree is also a partitioning BST. A Fano BST is a partitioning BST that satisfies an extra property, that of Definition 5.2, which states that given a partitioning BST, if for every internal node, the weights of its two children are as nearly equal as possible, that tree is a Fano BST. We show that if the input of the procedure updateFanoBST(...) is a Fano BST, the output is a Fano BST too.

Theorem 5.2. Let $\mathcal{T}(v) = \{t_1(v), \ldots, t_{2m-1}(v)\}$ be the Fano BST at time 'v', and $t_x(v)$ be the leaf of $\mathcal{T}(v)$ associated with the last processed symbol, $x[v]$. Suppose that $\mathcal{T}(v)$ and $t_x(v)$ are the parameters of procedure updateFanoBST(...). Then, the resulting tree, $\mathcal{T}(v+1)$, is a Fano BST.

Proof. Let $\mathcal{T}(v+1) = \{t_1(v+1), \ldots, t_{2m-1}(v+1)\}$ be the tree obtained after updating[9] $\mathcal{T}(v)$ at time 'v'. We have to prove that, for $u = 1, \ldots, m$, $t_{2u}(v+1)$ satisfies either

[9]Strictly speaking, $\mathcal{T}(v+1) = \mathcal{T}'$. But since the tree index is specifically included, we omit the "primed" notation, since there is no ambiguity.

(a), (b) or (c) of Definition 5.2.[10]

First of all, we observe that the procedure updateFanoBST(...) proceeds by incrementing, by unity, the weight of all the nodes in the path from $t_x(v)$ to $t_r(v)$, where $t_r(v)$ is the root of $\mathcal{T}(v)$, and $t_x(v)$ is the node associated with $x[v]$. Since $\mathcal{T}(v)$ is a Fano BST, we see that all the internal nodes of $\mathcal{T}(v+1)$, which are not in that path, are not modified, and hence they satisfy the conditions of Definition 5.2. We merely concern ourselves with the nodes that are modified by the procedure updateFanoBST(...).

Let $\pi_1, \ldots, \pi_\ell$ be the nodes in the path from $t_r(v)$ to $t_x(v)$. Starting from π_1, each node in the path, π_i, is verified by invoking the procedure checkShift(...). Let π_j and π_k be the left-most and the right-most leaves of the subtrees rooted at R_i and L_i respectively. With regard to its children's weights, we consider the following three mutually exclusive and exhaustive cases for π_i:

(*i*) $\tau_{L_i} < \tau_{R_i}$. In this case, since $\tau_{L_i} - \tau_{R_i} < 0$ and $\tau_k > 0$, the criterion function $\theta_2 < 0$, and hence no STR is performed. Furthermore, when we consider the criterion θ_1, it is easy to see that $\theta_1 > 0$ whenever $\tau_{R_i} - \tau_{L_i} > \tau_j$. In this scenario, an STL is performed on π_i, and τ_j is subtracted from τ_{R_i} and added to τ_{L_i}. Let $\pi_i = t_{2u}(v+1)$ for some integer u, $1 \leq u \leq m$. Since $\pi_j = t_{2u+1}$,

$$\tau_{R_u} - \tau_{L_u} \leq \tau_{2u+1} ,$$

and hence Condition (a) of Definition 5.2 is satisfied.

(*ii*) $\tau_{L_i} > \tau_{R_i}$. In this case, since $\tau_{R_i} - \tau_{L_i} < 0$, and $\tau_j > 0$, we see that the criterion function $\theta_1 < 0$, and subsequently an STL operation is *not* performed. On the other hand, we know that an STR operation is performed when $\theta_2 = \tau_{L_i} - \tau_{R_i} - \tau_k > 0$, which implies that $\tau_{L_i} - \tau_{R_i} > \tau_k$. Since the STR process requires that τ_k is subtracted from τ_{L_i}, we see that $\tau_{L_i} - \tau_{R_i} \leq \tau_k$ after the

[10] As per Definition 5.2, a Fano BST is a partitioning BST that satisfies certain constraints about the weights of the internal nodes and their children. To prove this theorem, we use the results of Lemmas 5.2, 5.3, 5.4, 5.5, 5.6, and 5.7, which state that after an STL or an STR operation is performed, the resulting tree satisfies the properties of a partitioning BST. Assuming that this is true, we prove that the resulting tree after the updating procedure satisfies the conditions of Definition 5.2.

STR operation. Let $\pi_i = t_{2u}(v+1)$ for some integer u, $1 \leq u \leq m$. This means that

$$\tau_{L_u} - \tau_{R_u} \leq \tau_{2u-1} ,$$

and hence Condition (b) of Definition 5.2 is satisfied.

(*iii*) $\tau_{L_i} = \tau_{R_i}$. This trivial case follows since it is equivalent to Condition (c) of Definition 5.2. In other words, the nearly-equal-probability property, fundamental in Fano coding, is satisfied.

From the three cases above, and the fact that the procedure checkShift(...) is recursively invoked for L_i and R_i, it is clear that either (a), (b) or (c) of Definition 5.2 is satisfied for all $t_{2u}(v+1)$, $u = 1, \ldots, m$, and hence $\mathcal{T}(v+1)$ is a Fano BST. □

We now study the lossless-compression properties of the Fano BST encoding and decoding algorithms. We show that encoding an input sequence using Algorithm **Fano_BST_Encoding**, into an encoded sequence, and decoding that sequence using Algorithm **Fano_BST_Decoding**, results into the original input sequence. This result is stated and proved in Theorem 5.3 given below. In the subsequent theorem, Theorem 5.4, we show that using Fano BST coding, on-line compression is achieved.

Theorem 5.3. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$, and the code alphabet $\mathcal{A} = \{0, 1\}$. Suppose that the input sequence $\mathcal{X} = x[1] \ldots x[M]$ is encoded by using Algorithm **Fano_BST_Encoding**, yielding an output sequence $\mathcal{Y} = y[1] \ldots y[R]$. If $\mathcal{Y}$ is decoded by using Algorithm **Fano_BST_Decoding** into $\mathcal{X}^* = x^*[1] \ldots x^*[T]$, then $\mathcal{X}^* = \mathcal{X}$ (i.e. $T = M$, and $x^*[i] = x[i]$, $\forall i$, $i = 1, \ldots, M$).

Proof. This result is proved by a double induction on the number of encoding steps. The outer loop of the induction is based on the encoding of $x[k]$ at the kth step, and the inner induction on the depth of the node visited in traversing the Fano BST in the encoding and the decoding.

First of all, we observe that at the kth encoding step, the encoding and the decoding are composed of two well-separated phases:

(*i*) Task of Encoder : Encode $x[k]$ using the Fano BST resulting from the $(k-1)^{th}$ step, $\mathcal{T}(k)$.

Task of Decoder : Decode the relevant encoded sequence into $x^*[k]$ using $\mathcal{T}(k)$.

The proof of this part is completed by an induction on the number of steps 'j' of the depth of the node visited in the Fano BST. We refer to this quantity as the "depth of the binary search".

(*ii*) Task of both Encoder and Decoder : Update $\mathcal{T}(k)$ yielding $\mathcal{T}(k+1)$. We observe that the steps involved in both the encoder and the decoder are identical, since they both invoke the procedure updateFanoBST(...), with identical parameters : $\mathcal{T}(k)$, the root of $\mathcal{T}(k)$, and the node associated with $x[k]$, $t_x(k)$.

The doubly-inductive proof on the number of encoding steps and the number of steps involved in the binary search follows.

Let $\pi_1(k), \ldots, \pi_{\ell_k}(k)$ be the nodes in the path from the root to $t_x(k)$, which is the node associated with $x[k]$. Note that ℓ_k is the length of the code word sent to the output at time 'k'.

Basis step for Outer Induction : Both the encoder and the decoder start with the same Fano BST, $\mathcal{T}(1)$, which is constructed by invoking the procedure Fano-BST(...) with identical parameters.

To prove that $x^*[1] = x[1]$, we show that at every step 'j' along the path of the binary search, the encoder and the decoder point to the same node in $\mathcal{T}(1)$, namely $\pi_j(1)$.

Basis step for $x[1]$: We shall prove that the encoder and the decoder point to the same node for $j = 1$.

Let i be the position of $x[k]$ in $\mathcal{T}(1)$ from left to right (counting the leaves only), and $\pi_j(1)$ be the current pointer. Both the encoder and the decoder start pointing to the root of $\mathcal{T}(1)$, $\pi_1(1)$. We identify two mutually exclusive and exhaustive cases for i.

(a) $i < \frac{k_1(1)+1}{2}$: This implies that $2i - 1 < k_1(1)$, a '0' is sent to the output, and the current pointer is set to $L_n(1) = \pi_2(1)$. The decoder receives a '0', and moves the current pointer to $L_n(1) = \pi_2(1)$.

(b) $i \geq \frac{k_1(1)+1}{2}$: Here, $2i - 1 \geq k_1(1)$. In this case, a '1' is sent to the output, and the current pointer is moved to $R_n(1) = \pi_2(1)$. The same '1' is received by the decoder, and the current pointer to $\pi_1(1)$ is also moved to $R_n(1) = \pi_2(1)$.

Inductive hypothesis for $x[1]$: Let us suppose that the result holds along the depth of the binary search for $j = u$, $u > 1$. This implies that in both, the encoder and the decoder, the current pointer is pointing to the same node in $\mathcal{T}(1)$, $\pi_u(1)$.

Inductive step for $x[1]$: The induction is now completed by showing that the result holds for $j = u + 1$. From the inductive hypothesis, in both the encoder and the decoder, the current pointer points to the same node, $\pi_u(1)$. As in the basis step, we have two mutually exclusive and exhaustive cases for i:

(a) $i < \frac{k_u(1)+1}{2}$: This implies that $2i - 1 < k_u(1)$, and hence a '0' is sent to the output, and the current pointer is moved to the left child of $\pi_u(1)$, $L_u(1) = \pi_{u+1}(1)$. The decoder receives a '0', and moves the current pointer to $L_u(1) = \pi_{u+1}(1)$ as well.

(b) $i \geq \frac{k_u(1)+1}{2}$: This inequality is equivalent to $2i - 1 \geq k_u(1) + 1$, which implies that a '1' is sent to the output, and the current pointer points to $R_u(1) = \pi_{u+1}(1)$. The symbol '1' is now received by the decoder, which also moves the current pointer to $R_u(1) = \pi_{u+1}(1)$.

As we see, for all $j \geq 1$, the current pointer points to the same node of $\mathcal{T}(1)$ in both the encoder and the decoder. Clearly, when $\pi_j(1)$ is a leaf, the decoder is pointing to the node of $\mathcal{T}(1)$ associated with $x^*[1]$, which is thus necessarily $x[1]$.

On the other hand, both the encoder and the decoder update $\mathcal{T}(1)$ by invoking the procedure updateFanoBST(...) with identical parameters, obtaining the same Fano BST, $\mathcal{T}(2)$. The *basis* step of the outer induction is thus proven.

We now continue the outer level of the induction which involves the processing of $x[v]$.

Inductive hypothesis for Outer Induction: At this stage of the proof, we assume that the outer level induction result holds for $k = v$, where $v > 1$. This implies that the encoder processes $x[v]$ using $\mathcal{T}(v)$, and the decoder successfully recovers $x^*[v] = x[v]$ using $\mathcal{T}(v)$ as well. They both update $\mathcal{T}(v)$ by invoking the procedure updateFanoBST(...) with identical parameters, obtaining the same Fano BST, $\mathcal{T}(v+1)$.

Inductive step for Outer Induction: We now have to prove that the encoder encodes $x[v+1]$, and the decoder recovers exactly the same symbol, $x^*[v+1] = x[v+1]$. We prove this by an induction on the value of 'j', the depth of the binary search.

Basis step for $x[v+1]$: We have to prove that both the encoder and the decoder point to the same node in $\mathcal{T}(v+1)$. Again, as before, we let i be the position in $\mathcal{T}(v+1)$ from left to right (counting the leaves only). Both the encoder and the decoder start with the current pointer pointing to the root of $\mathcal{T}(v+1)$. We thus have two mutually exclusive and exhaustive cases for i :

(a) $i < \frac{k_1(v+1)+1}{2}$: In this case, we see that $2i - 1 < k_1(v+1)$, which implies that the encoder sends a '0' to the output, and sets the current pointer to $L_1(v+1) = \pi_2(v+1)$. The decoder thus receives a '0', and sets the current pointer to $L_1(v+1) = \pi_2(v+1)$ as well.

(b) $i \geq \frac{k_1(v+1)+1}{2}$: This implies that $2i - 1 \geq k_1(v+1)$. A '1' is sent to the output, and the current pointer is moved to $R_1(v+1) =$

$\pi_2(v+1)$. The same symbol '1' is received by the decoder, which moves the current pointer to $R_1(v+1) = \pi_2(v+1)$ too.

This proves the basis step for *this* inner induction.

Inductive hypothesis for $x[v+1]$: We assume that the result holds for $j = u$, $u > 1$. This implies that both the encoder and the decoder have $x^*[1] = x[1], \ldots, x^*[v] = x[v]$, and that the current pointer points to the same node in $\mathcal{T}(v+1)$, $\pi_u(v+1)$.

Inductive step for $x[v+1]$: To complete the induction on the number of steps of the binary search, we have to prove that in both the encoder and the decoder, the current pointer points to the same node of $\mathcal{T}(v+1)$ for $j = u+1$.

From the inductive hypothesis, in both the encoder and the decoder, the current pointer points to the same node in $\mathcal{T}(v+1)$, $\pi_u(v+1)$. Again, we have two mutually exclusive and exhaustive cases for i :

(a) $i < \frac{k_u(v+1)+1}{2}$: In this case, $2i - 1 < k_u(v+1)$, which implies that a '0' is sent to the output, and the current pointer is set to $L_u(v+1) = \pi_{u+1}(v+1)$. Consequently, a '0' is received by the decoder, which also moves the current pointer to $L_u(v+1) = \pi_{u+1}(v+1)$.

(b) $i \geq \frac{k_u(v+1)+1}{2}$: This implies that $2i-1 \geq k_u(v+1)$, and hence a '1' is sent to the output, and the current pointer is set to $R_u(v+1)$. On the decoder's side, a '1' is received, and the current pointer is set to $R_u(v+1) = \pi_{u+1}(v+1)$ as well.

We have thus proved that for all $j \geq 1$, the current pointer points to the same node of $\mathcal{T}(v+1)$ in both the encoder and the decoder. Consequently, when $\pi_j(v+1)$ is a leaf, both the encoder and the decoder will point to the same node of $\mathcal{T}(v+1)$ associated with the symbol $x^*[v+1]$, and hence the outer inductive proof holds for $k \geq 1$.

Therefore, we see that the result holds for all $v \geq 1$, and hence the outer inductive proof on the number of encoding steps is also completed. The result thus follows. □

Theorem 5.4. Consider the assumptions stated in Theorem 5.3. Encoding $\mathcal{X}$ into $\mathcal{Y}$ using Algorithm **Fano_BST_Encoding**, and decoding $\mathcal{Y}$ into $\mathcal{X}$ following the procedure of Algorithm **Fano_BST_Decoding** yield *on-line compression.*

Proof. From the inductive proof of Theorem 5.3, we see that each symbol encoded from the input, $x[k]$, is simultaneously decoded into $x^*[k] = x[k]$. This is done by decoding its code word $y[j] \ldots y[j + \ell_k]$, where ℓ_k is the length of the code word of $x[k]$, in a prefix manner. Consequently, encoding by using Algorithm **Fano_BST_Encoding** and decoding by using algorithm **Fano_BST_Decoding** achieve *on-line compression* at each individual bit processed. □

The procedure updateFanoBST(...) checks, for every internal node in the path from the root to the leaf associated with the symbol occurred so far, if the nearly-equal-probability property is satisfied. This takes a small amount of time, $O(\log n)$. However, if an STL or an STR operation is performed on a non-two-leaf internal node, then its children have also to be evaluated to see if a shift operation must be performed on these nodes too. This may take a long time[11], specially, for the root node or the internal nodes close to it. The aim of using Fano BST coding is that, for stationary sources, fewer changes in the Fano BST are expected after encoding a large number of source symbols, avoiding the code word reconstruction required in the greedy adaptive Fano coding discussed in Chapter 4.

In this regard, we show that when the probabilities of the source alphabet symbols are unknown, the Fano BST, which is maintained by invoking the procedure updateFanoBST(...), asymptotically converges to the Fano tree constructed by following the procedure of Algorithm **Static_Fano_Encoding** (see Section 2.4.3) when the probabilities of the source symbols, $\{p_1, \ldots, p_m\}$, are known *a priori*. This implies that, asymptotically, no extra information (or computation) is required to maintain the Fano BST. The result is formalized below in the following theorem.

[11] Up to $O(\log n)$ steps are required to walk down from the root node to the leaf associated with $x[k]$. For each of these steps, up to $O(n)$ steps are required by procedure chekcShift(...). Consequently, up to $O(n \log n)$ steps are required.

Theorem 5.5. Consider a stationary source whose alphabet is $\mathcal{S} = \{s_1, \ldots, s_m\}$, and their probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq \ldots \geq p_m$. If $\eta(k) = \{\eta_1(k), \ldots, \eta_m(k)\}$ are the frequency counters of $\{s_1, \ldots, s_m\}$, the Fano BST maintained using the procedure updateFanoBST(...) asymptotically converges to the static Fano tree, where the latter is the tree that would have been obtained if the occurrence probabilities are known *a priori*.

Proof. Let $\mathcal{X} = x[1] \ldots x[M]$ be the input sequence which is being encoded by Algorithm **Fano_BST_Encoding**. First of all, observe that the leaves of the Fano BST, which is adaptively maintained by procedure updateFanoBST(...), are associated with the source symbols $\{s_1, \ldots, s_m\}$, where $p_1 \geq \ldots \geq p_m$.

Additionally, we know that the source is stationary, and so the symbol $x[k]$ is processed with a fixed (but unknown) probability p_i. The symbol s_i is the one being used to update the estimates of the probabilities of the leaves in the Fano BST. Consequently, by the law of large numbers, the estimated probabilities of the leaves, $\{\hat{p}_1, \ldots, \hat{p}_m\} = \{\frac{\eta_1(k)}{m+k}, \ldots, \frac{\eta_m(k)}{m+k}\}$, given by the ratios of the counters to the number of symbols processed, asymptotically converge to the true but unknown probabilities.

We now have to argue the asymptotic equivalence of the static Fano tree and the corresponding Fano BST. The static Fano tree is created using Algorithm **Static_Fano_Encoding**, which partitions $\mathcal{S}$ and $\mathcal{P}$ into two sublists such that the nearly-equal-probability property is satisfied. The procedure is recursively invoked to each of these two sublists, until a sublist with a single symbol is obtained, in which case, the recursion stops.

From the procedure updateFanoBST(...), we see that the updating procedure starts from the root of the tree, walking down towards the leaves, while simultaneously checking that the nearly-equal-probability property is satisfied.

Consider the adaptive updating procedure being performed with t_i, the root of the Fano BST. The leaves of the left child, L_i, happen to be $s_1, \ldots, s_j$, and the leaves of the right child, R_i, are $s_{j+1}, \ldots, s_m$, where $\sum_{l=0}^{j} \hat{p}_l$ and $\sum_{l=j+1}^{m} \hat{p}_l$ are as nearly-equal as possible. This is ensured by the updating rule of the procedure updateFanoBST(...), which checks that τ_{L_i} and τ_{R_i} satisfy the nearly-equal-probability property. This procedure is recursively invoked for L_i and R_i until the leaves are reached.

The theorem follows by observing that the list maintained by the static Fano encoding is **identical** to the asymptotic list that would have been obtained by traversing the *leaves* of the Fano BST in a left-to-right manner. The only advantage, however, is that in the Fano BST, we do not have to traverse the "list" of the leaves to obtain the partitioning. The sum of the probabilities of the sublists reside in the corresponding τ values of the parents of these leaves, all the way up to the root. The result is thus proven. □

A direct consequence of the above theorem is the following corollary which is obvious, and therefore, merely stated.

Corollary 5.1. The number of updates in the Fano BST asymptotically tend to zero.

We conclude this section by observing that the result of Theorem 5.5 could also be formalized for *any* other adaptive encoding method that uses an analogous frequency-counter updating rule. We thus believe that similar strategies are applicable for other methods described in Chapter 2, namely Huffman coding, arithmetic coding, Shannon's method, etc. These remain open problems which we are currently investigating.

5.4 Empirical Analysis

In order to analyze the performance of the adaptive Fano coding using BSTs, we have performed some empirical tests on the files of the well-known benchmarks, namely the Calgary corpus and the Canterbury corpus. First of all, we have tested the adaptive Fano coding using BSTs and the greedy adaptive Fano coding presented in Chapter 4 with regard to speed and compression efficiency.

On the other hand, to analyze the variability of the Fano BST during the encoding, we have performed an empirical analysis on the number of input symbols that require shift operations, and the total number of shifts of the entire encoding process.

5.4.1 Speed and Compression Efficiency

To analyze the speed and compression efficiency of the adaptive Fano coding using BSTs, we report the results of running these techniques on the files of the Calgary

File Name	$l_{\mathcal{X}}$ (bytes)	Greedy Fano			Fano BST		
		ρ (%)	T/Enc. (sec.)	T/Dec. (sec.)	ρ (%)	T/Enc. (sec.)	T/Dec. (sec.)
bib	111,261	34.21	1.31	0.86	34.21	1.32	1.24
book1	768,771	42.83	6.97	5.10	42.84	4.00	3.55
book2	610,856	39.47	6.04	4.41	39.48	4.14	3.81
geo	102,400	28.66	2.24	0.85	28.65	1.54	1.47
news	377,109	34.45	4.58	2.88	34.46	2.76	2.57
obj1	21,504	24.20	0.49	0.20	23.96	0.86	0.80
obj2	246,814	21.21	5.39	2.43	21.16	2.16	2.07
paper1	53,161	36.80	0.57	0.38	36.80	1.06	1.00
progc	39,611	33.64	0.50	0.28	33.62	1.00	0.93
progl	71,646	39.20	0.71	0.52	39.21	1.65	1.59
progp	49,379	37.80	0.54	0.35	37.80	1.16	1.13
trans	93,695	30.04	1.41	0.77	30.03	1.11	1.06
Total	2,547,207	36.68	30.75	19.03	36.64	22.76	21.22

Table 5.1: Speed and compression ratio for the Fano BST coding and the greedy adaptive Fano method tested on files of the Calgary corpus.

corpus and the Canterbury corpus. We have also run the greedy adaptive Fano coding presented in Chapter 4 on the same benchmarks. The results obtained from the tests are shown in Table 5.1 for the files of the Calgary corpus, and in Table 5.2 for the files of the Canterbury corpus.

The content of the two tables is as follows. The first column contains the name of the file. The second column, labeled $l_{\mathcal{X}}$, represents the size (in bytes) of the original file. The next columns correspond to the speed and compression ratio for the adaptive greedy Fano coding (Greedy Fano), and the adaptive Fano coding using BSTs (Fano BST). Each of the two groups of three columns contains the compression ratio, ρ, as calculated in (2.18), the time (in seconds) required to compress the file, and the time (in seconds) needed to recover the original file.

For the files of the Calgary corpus, we observe that GreedyFano and Fano BST achieve similar compression ratios. From the weighted average (the row labeled "Total"), we observe that Greedy Fano compresses slightly more (but only 0.04%) than

File Name	$l_{\mathcal{X}}$ (bytes)	Greedy Fano			Fano BST		
		ρ (%)	T/Enc. (sec.)	T/Dec. (sec.)	ρ (%)	T/Enc. (sec.)	T/Dec. (sec.)
alice29.txt	148,481	42.59	1.20	0.99	42.59	1.44	1.33
asyoulik.txt	125,179	39.05	1.25	0.89	39.06	1.31	1.23
cp.html	24,603	33.19	0.30	0.19	33.19	0.88	0.82
fields.c	11,150	34.92	0.14	0.08	34.89	0.52	0.49
grammar.lsp	3,721	37.29	0.05	0.02	37.24	0.32	0.29
kennedy.xls	1,029,744	54.66	11.72	5.76	54.81	4.50	3.86
lcet10.txt	419,235	41.72	3.83	2.84	41.72	2.41	2.19
plrabn12.txt	471,162	43.32	4.20	3.08	43.32	2.33	2.16
ptt5	513,216	79.16	1.68	1.29	79.16	1.86	1.60
sum	38,240	31.50	0.66	0.30	31.42	0.92	0.88
xargs.1	4,227	34.81	0.06	0.03	34.79	0.42	0.38
Total	2,788,958	53.33	25.09	15.47	53.38	16.91	15.23

Table 5.2: Speed and compression ratio obtained after running the greedy adaptive Fano coding and the Fano BST coding on files of the Canterbury corpus.

Fano BST. In fact, this behavior is reasonable since they are expected to achieve the same compression ratio, as they use the principles of the Fano coding implemented with different structures. In terms of compression speed, Greedy Fano is faster than Fano BST on small files, namely *obj1*, *paper1*, *progc*, *progl*, and *progp*. On large files, Fano BST performs faster that Greedy Fano. This is expected, since for large files, less changes are expected in the Fano BST. This is also graphically analyzed in the next subsection. In the decompression stage, Greedy Fano is marginally faster than Fano BST, mainly in the small files mentioned above.

On the Canterbury corpus benchmark, the compression ratio obtained after running Greedy Fano and Fano BST are also, as expected, similar. We observe that Fano BST obtains compression ratios slightly higher (only 0.05%) than Greedy Fano. In regard to speed of compression, the behavior is similar to the Calgary corpus test. On small files, Greedy Fano is faster than Fano BST, and on large files, except for *ptt5*, Fano BST is faster. In fact, the reason of why Greedy Fano is faster on *ptt5* is due to the fact that a high compression ratio is achieved, and the input file contains only a few different

File Name	$l_{\mathcal{X}}$ (bytes)	$\ell_{\mathcal{Y}}$ (bits)	γ_b (steps)	φ_b (%)	F_b (steps)	ζ_s (shifts)	ζ_n (shifts)
bib	111,261	585,546	1,168	1.05	95.26	96,739	0.165
book1	768,771	3,515,626	1,032	0.13	744.93	134,456	0.038
book2	610,856	2,957,627	1,443	0.24	423.32	184,894	0.063
geo	102,400	584,536	2,846	2.78	35.98	132,732	0.227
news	377,109	1,977,284	1,589	0.42	237.32	144,728	0.073
obj1	21,504	130,810	1,964	9.13	10.95	92,847	0.710
obj2	246,814	1,556,609	3,168	1.28	77.91	134,129	0.086
paper1	53,161	268,802	937	1.76	56.74	94,709	0.352
progc	39,611	210,354	1,081	2.73	36.64	97,648	0.464
progl	71,646	348,414	1,193	1.67	60.06	136,627	0.392
progp	49,379	245,697	965	1.95	51.17	107,752	0.439
trans	93,695	524,451	1,054	1.12	88.89	91,488	0.174

Table 5.3: Empirical results for the variability of the Fano BST obtained after running the adaptive Fano coding using BSTs on files of the Calgary corpus.

symbols, which makes the BST similar to a list. Consequently, its behavior is similar to that of the Greedy Fano, with the burden of maintaining a complex structure, the Fano BST. Observe that to compress the file *kennedy.xls*, Greedy Fano takes more than twice and half as much time as that of Fano BST. The two methods decompress the Canterbury benchmark in approximately the same number of seconds.

5.4.2 Variability of the Fano BST

In this section, we analyze how the structure of the Fano BST varies during the encoding process. To proceed with the analysis, we have tested the adaptive Fano coding using BSTs on the files of the Calgary corpus and the Canterbury corpus. We analyze the variability of the Fano BST using two different measures: the number of encoding steps with shift operations, and the total number of shifts in the entire encoding process. An encoding step involves reading a symbol from the input sequence and generating its code word. The empirical results obtained are shown in Tables 5.3 and 5.4.

As usual, the first and second columns contain the file name and the size (in bytes) of the original file. The third column contains the size (in bits) of the compressed file,

File Name	$l_{\mathcal{X}}$ (bytes)	$\ell_{\mathcal{Y}}$ (bits)	γ_b (steps)	φ_b (%)	F_b (steps)	ζ_s (shifts)	ζ_n (shifts)
alice.txt	148,481	681,903	910	0.61	163.17	97,795	0.143
asyoulik.txt	125,179	610,304	905	0.72	138.32	23,260	0.038
cp.html	24,603	131,498	841	3.42	29.25	86,699	0.659
fields.c	11,150	58,080	714	6.40	15.61	58,760	1.012
grammar.lsp	3,721	18,683	462	12.42	8.05	38,320	2.051
kennedy.xls	1,029,744	3,722,970	3,508	0.34	293.54	155,813	0.042
lcet10.txt	419,235	1,954,593	1,038	0.25	403.89	109,030	0.056
plrabn12.txt	471,162	2,136,386	788	0.17	597.92	87,801	0.041
ptt5	513,216	855,435	1,349	0.26	380.44	114,144	0.133
sum	38,240	209,802	1,832	4.79	20.87	95,747	0.456
xargs.1	4,227	22,052	481	11.38	8.78	45,876	2.080

Table 5.4: Variability of the Fano BST during the encoding process on the files of Canterbury corpus.

$\ell_{\mathcal{Y}}$. The next columns are detailed below:

γ_b : The total number of encoding steps in which at least one shift operation was performed.

φ_b : The percentage of encoding steps with at least one shift operation, calculated as follows:

$$\varphi_b = \frac{\gamma_b}{l_{\mathcal{X}}} 100 . \tag{5.19}$$

F_b : The frequency of steps involving shift operations. This value indicates how often an encoding step contains at least one shift operation, and is calculated as follows:

$$F_b = \frac{l_{\mathcal{X}}}{\gamma_b} . \tag{5.20}$$

ζ_s : The total number of shift (STL or STR) operations performed in the entire encoding process.

ζ_n : The number of shift operations per node visited when generating the code words, calculated as follows:

$$\zeta_n = \frac{\zeta_s}{\ell_y}. \tag{5.21}$$

From the results shown in the tables, we observe that although the total number of steps with shift operations are smaller for small files, the difference for large files is not proportional. For example, for the file *paper1*, $\gamma_b = 937$, and for the file *bib* (which is more than twice as large as *paper1*) $\gamma_b = 1,168$. This behavior is reflected in the percentage of steps with shift operations, φ_b. We see that for large files, such as *book1* and *book2*, φ_b is significantly smaller, 0.13 and 0.24 respectively. Conversely, φ_b is large for small files, such as *paper1*, for which $\varphi_b = 9.13$ (the highest value of the Calgary corpus test suite). For the files of the Canterbury corpus, φ_b is also large, 12.42 and 11.38 for small files, namely *grammar.lsp* and *xargs.1* respectively. Conversely, φ_b is small for large files.

With regard to the frequency of steps with shifts, F_b is small for small files and vice versa, for the files of the Calgary corpus and the Canterbury corpus. This behavior is as expected, since fewer changes in the Fano BST are likely to occur after some time, when the estimated probabilities converge to the actual probabilities of the source symbols. When analyzing the number of shifts per node visited during the encoding, we observe that ζ_n is large for small files and vice versa. The behavior observed in these empirical results clearly corroborate the theoretical analysis presented in Section 5.3.7.

5.5 Graphical analysis

In order to have a better perspective of the empirical results discussed in the previous section, we present a graphical analysis of these results here. For this analysis we use the empirical results shown in Tables 5.3 and 5.4, which correspond to the simulations that we have run for the files of the Calgary corpus and the Canterbury corpus respectively. These tables contain various pieces of information related to the use of the Fano BST during the encoding process, for the two corpuses.

The first graph, which we show in Figure 5.10, plots the percentage of encoding steps that invoke at least *one* shift operation, φ_b. This figure has been drawn by recording the cumulative results over all the files of the Calgary corpus. The x-axis represents the size of the input file in bytes, $l_{\mathcal{X}}$, and the y-axis is the value of φ_b, which is calculated as in (5.19). Observe that the value of φ_b is very high for small files, and very small for large files. For example, φ_b is above 9% for *obj1* (the smallest file in the Calgary corpus), and approximately 0.13 for the file *book1*, which is the largest file of the Calgary corpus. Except for the "unexpected" peak observed near the $l_{\mathcal{X}} = 100$Kb boundary, which corresponds to the file *geo*, we observe that the value of φ_b decreases monotonically with the size of the files, and asymptotically tends to 0. This behavior is reasonable and is entirely consistent with the results of Theorem 5.5, from which we see that the estimated probabilities become closer to the actual probabilities of the symbols as the number of encoded symbols increases.

A similar plot of φ_b for the files of the Canterbury corpus is shown in Figure 5.11. From this plot, we again see that for small files (e.g. *grammar.lsp*) the percentage of steps with shift operations is large, and for big files, such as *kennedy.xls*, the value of φ_b is relatively small. This again corroborates the result of Theorem 5.5.

Another important measurement demonstrating the properties of the Fano BST is ζ_n, the number of shift operations per node visited when generating the output code words. As mentioned earlier, this quantity is calculated as in (5.21). A plot of ζ_n for the Fano BST constructed for the files of the Calgary corpus is shown in Figure 5.12. As in the case of φ_b, the value of ζ_n is relatively large for small compressed files, such as *obj1*, and very small for large files such as *kennedy.xls*, for which $\zeta_n = 0.056$. From

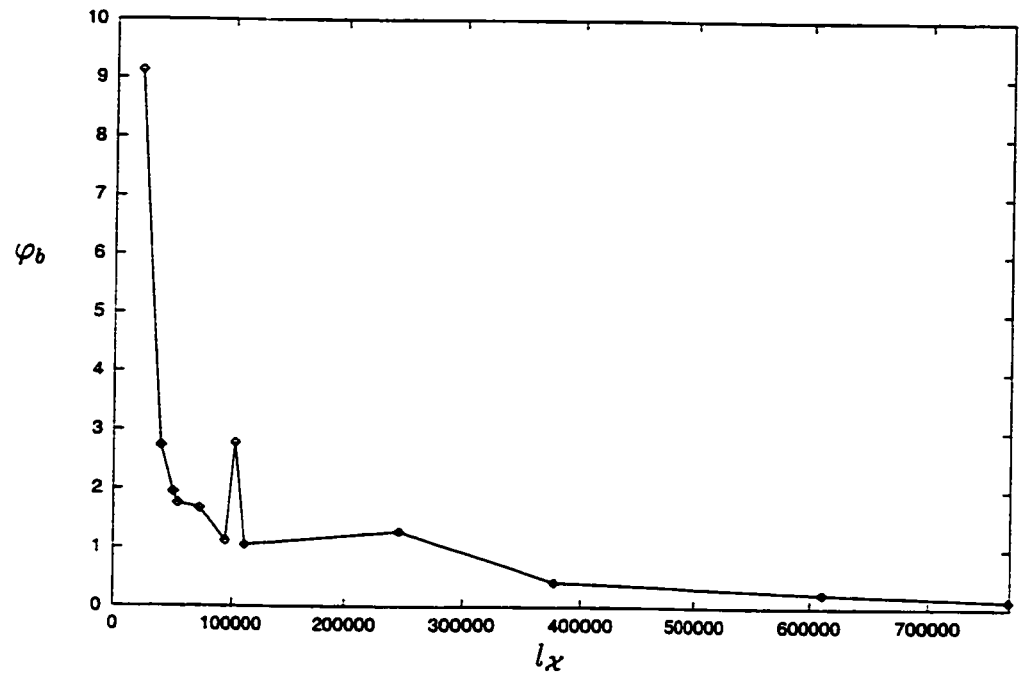


Figure 5.10: Plot of the percentage of encoding steps with at least one shift operation for the files of the Calgary corpus.

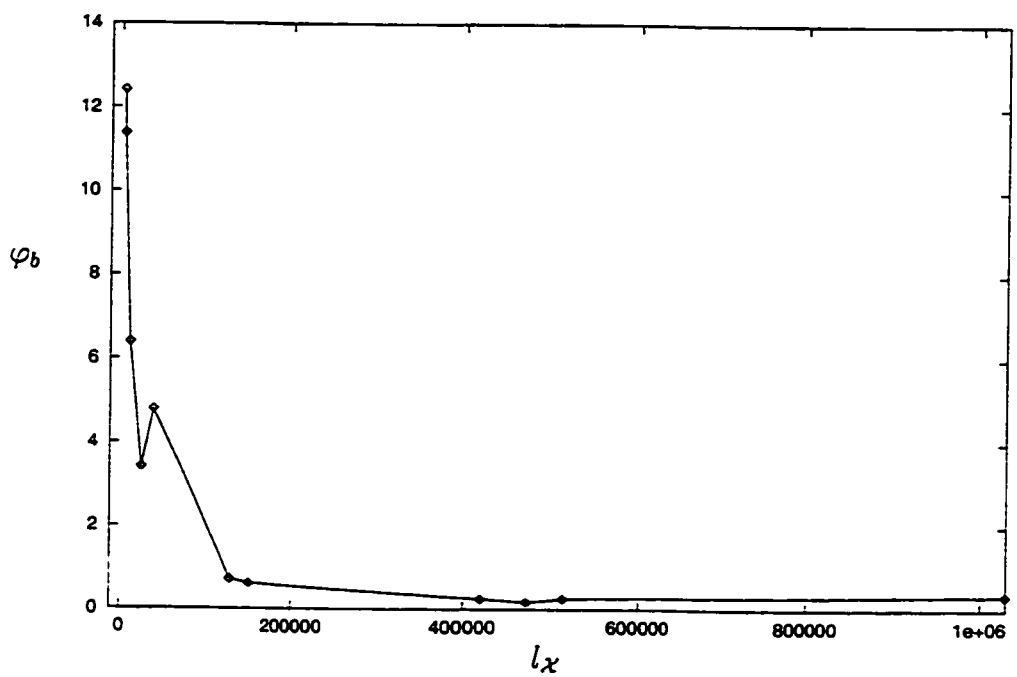


Figure 5.11: Percentage of encoding steps with shift operations vs. number of encoding steps for the files of the Canterbury corpus.

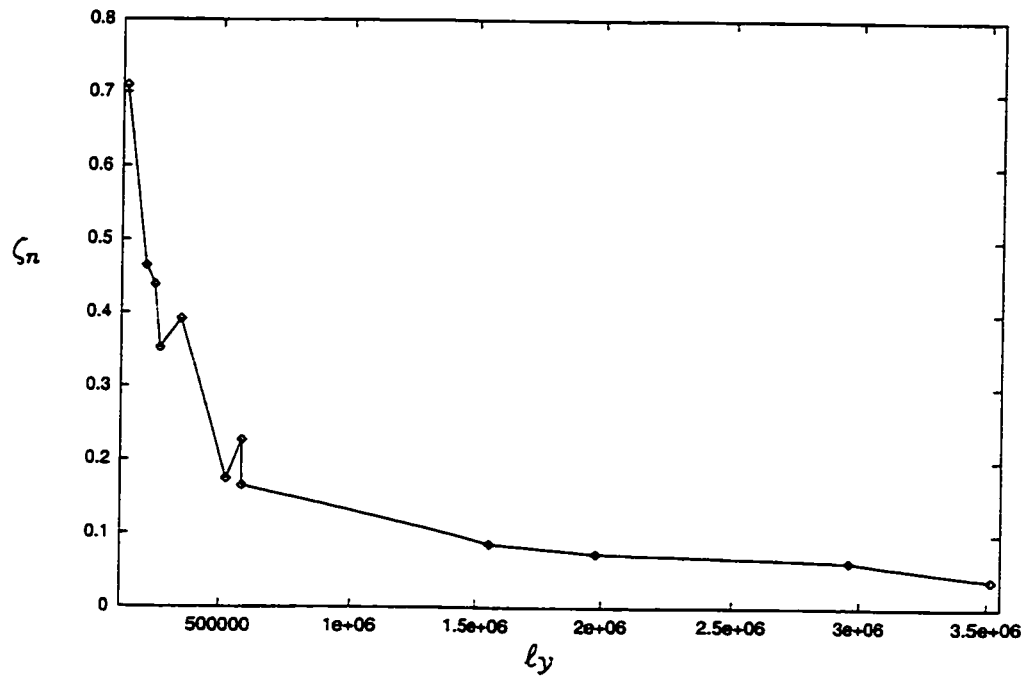


Figure 5.12: Number of shift operations per node visited in the entire encoding process for the files of the Calgary corpus.

the overall appearance of the plot, we also observe that ζ_n asymptotically converges to 0 as the number of bits (or nodes visited in the Fano BST) increases indefinitely. A similar plot for the files of the Canterbury corpus is shown in Figure 5.13, from which we again observe the same behavior, namely that the number of shifts per node visited is very small for large files such as *kennedy.xls*, for which $\zeta_n = 0.056$.

In order to enhance the visualization of the empirical results that we have obtained from our simulations on the files of the Calgary corpus and the Canterbury corpus, we have plotted another index obtained from updating the Fano BST. This index, $\hat{q}(n)$, which is the estimated probability of change in the Fano BST at time 'n', is calculated as the normalized number of shifts performed in a "window" of n_w bits sent to the output, normalized by the window width. In order to obtain a better visualization of the plot, we have set n_w to $\frac{\ell_y}{500}$, and hence, 500 points are included in the plot for each of the files tested. One such plot is shown in Figure 5.14, which corresponds to the estimated probability of shifts in the Fano BST during the encoding of the file *book1* of the Calgary corpus. Observe that the probability of change is very high

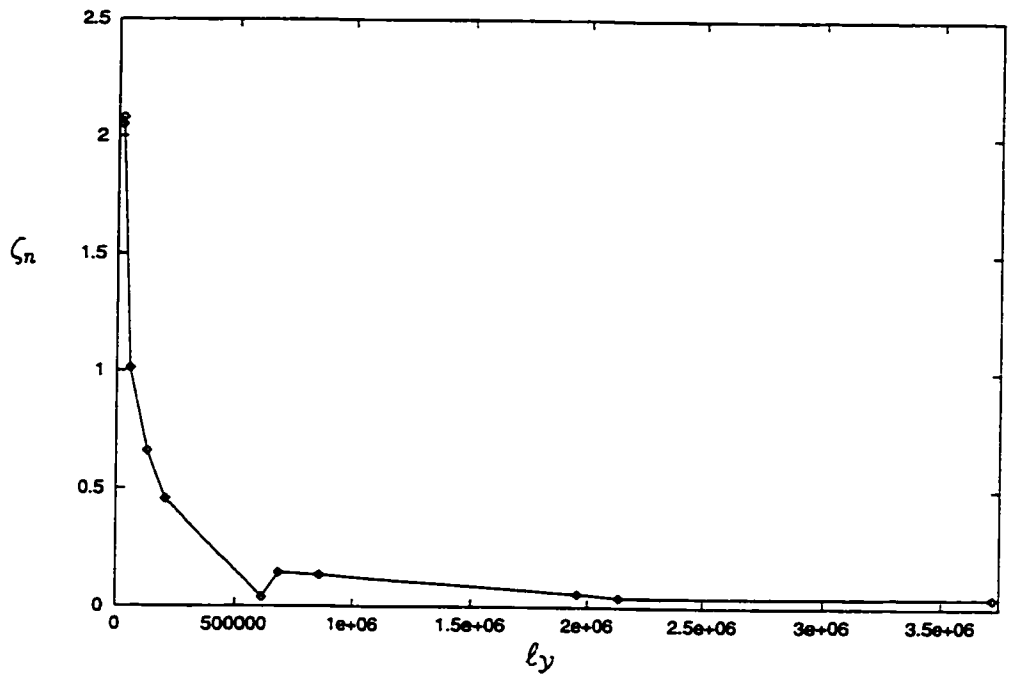


Figure 5.13: Plot of the number of shift operations per node visited vs. size of the output (in bits) for the files of the Canterbury corpus.

at the very beginning of the encoding process (e.g. $n \leq 1000$). This probability decreases monotonically and tends to 0 as the number of bits processed increases. A similar plot for the file *paper1* of the Calgary corpus is shown in Figure 5.15. Although the estimated probability of shifts in the Fano BST fluctuates, we again observe that this probability asymptotically tends to 0 as the number of encoded bits tends to 0. This behavior also confirms the results of Theorem 5.5, which states that the estimated probabilities of the input symbols converge to the true values as the number of symbols encoded increases indefinitely, and as this takes place, a lesser number of changes are expected in the Fano BST.

Similar plots for the estimated probability of shift per node for the files *cp.html* and *kennedy.xls* of the Canterbury corpus are depicted in Figures 5.16 and 5.17 respectively. Again, we observe that the estimated probability of change is large when a small amount of bits have been processed (as in the case of the file *cp.html*). A smaller variation is observed when a large number of bits have been encoded. Finally, the probability of change tends to 0 as the number of bits in the output tends to infinity, which occurs

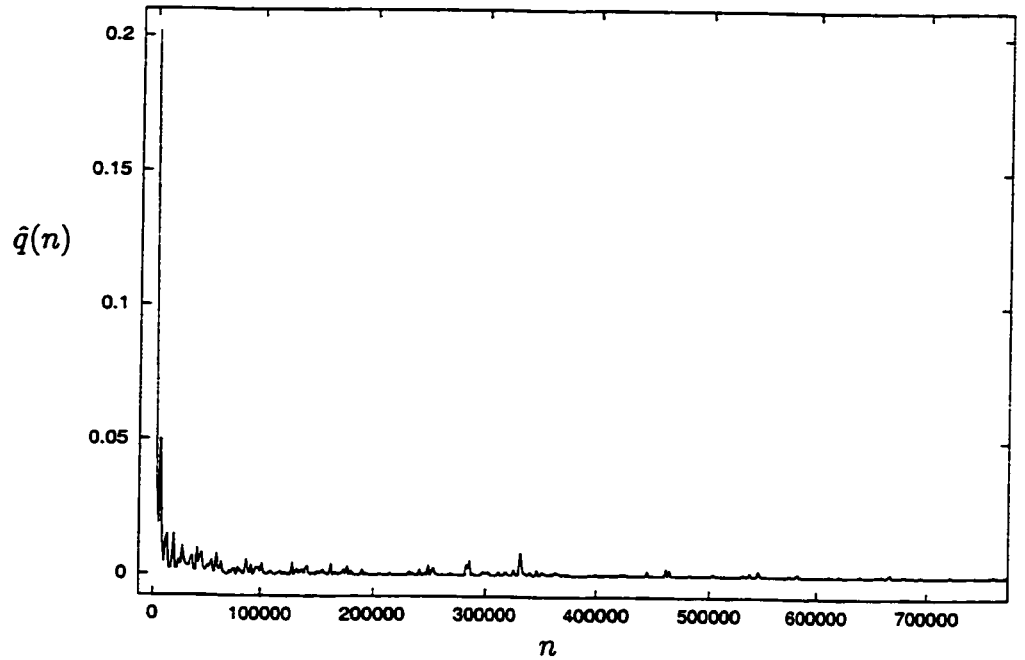


Figure 5.14: Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *book1* which was obtained from the Calgary corpus.

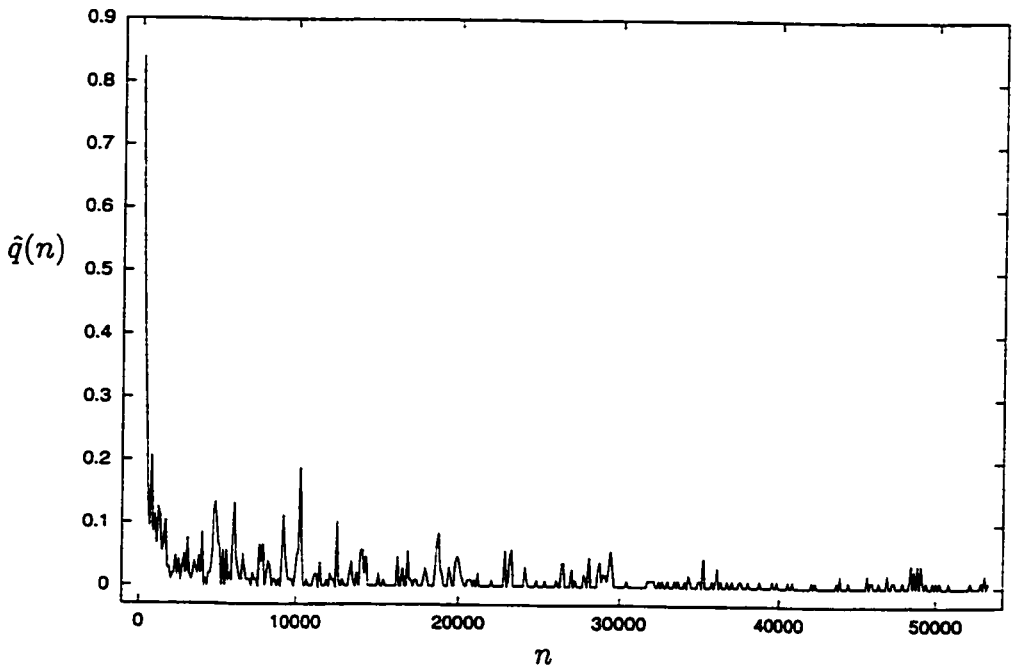


Figure 5.15: Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *paper1* which was obtained from the Calgary corpus.

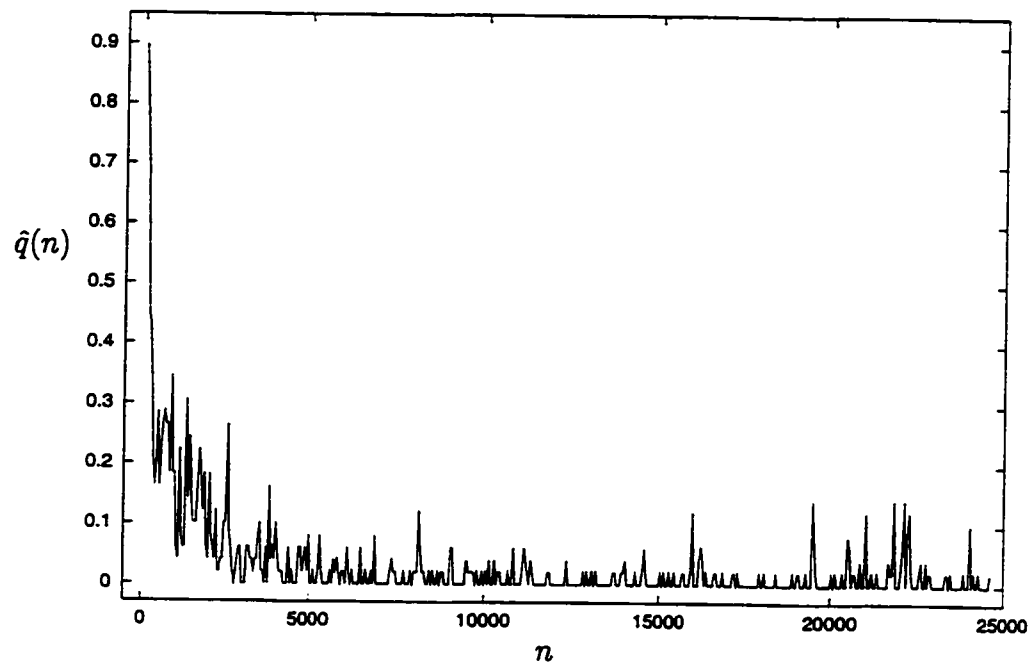


Figure 5.16: Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *cp.html* which was obtained from the Canterbury corpus.

when the estimated probabilities asymptotically tend to the actual probabilities of the input symbols.

5.6 Conclusions

In this chapter, we have introduced the adaptive Fano coding that uses a new structure, the Fano BST. This structure is based on a more fundamental data structure that we have defined, namely the partitioning BST. The latter is a BST in which the source alphabet symbols are incorporated in such a way that their location can be determined using their indices as the "keys". We have formally defined the Fano BST, which incorporates the required Fano property, namely the nearly-equal-probability property, characteristic of the Fano coding.

In order to maintain a Fano BST while encoding, we have introduced a heuristic that allows us to ensure, for each *non-two-leaf internal node*, that the nearly-equal-probability property is satisfied. The updating procedure that converts a partitioning

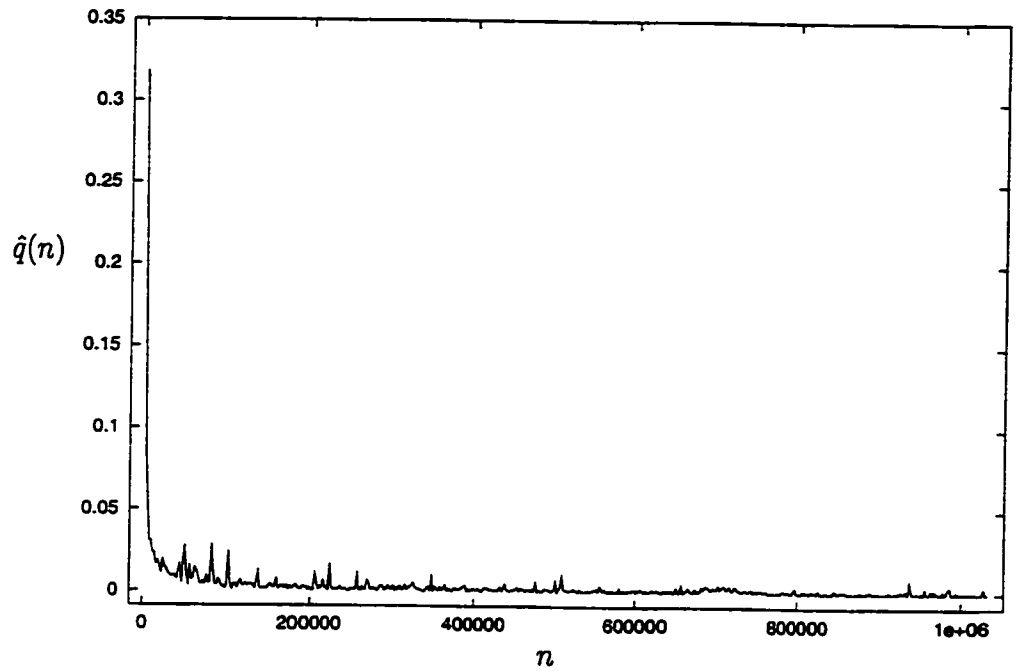


Figure 5.17: Plot of the estimated probability of shift at each encoding step, $\hat{q}(n)$, for the file *kennedy.xls* which was obtained from the Canterbury corpus.

BST into a Fano BST uses the latter heuristic and two new *tree*-based operators, namely the Shift-To-Left (STL) operator and the Shift-To-Right (STR) operator. For these two operators, we have identified three mutually exclusive scenarios in which they can be applied. We have provided the formal rules that implement these operators, as well as the respective scenarios for their validity. We have also introduced the corresponding encoding and decoding algorithms that invoke the corresponding updating procedures.

By means of a rigorous theoretical analysis, we have shown the correctness of the updating procedure, i.e. that it receives a partitioning BST, and using the tree-based operators mentioned above, outputs a Fano BST. We have also shown the validity of the encoding and decoding algorithms, which achieve lossless compression, and simultaneously maintain a Fano BST. Using these algorithms, on-line encoding/decoding can be attained, which is useful in networking and real-time applications such as LANs, WANs, the internet, e-mail, e-commerce, ftp, audio and video streaming, etc.

We have also analyzed the variability of the Fano BST during the encoding/decoding

process. We have shown that when dealing with stationary sources, the Fano BST asymptotically converges to the static Fano tree, and subsequently, fewer changes are required to maintain the "nearly-optimal" Fano BST.

We have tested our encoding and decoding algorithms on files of the well-known benchmarks, the Calgary corpus and the Canterbury corpus. Our empirical results show that the adaptive Fano coding using BSTs and the greedy adaptive Fano achieve similar compression ratios. In terms of encoding/decoding speed, the former is much faster than the latter in the encoding phase, and they achieve approximately the same speed in the decoding phase.

We have empirically tested how the Fano BST changes during the encoding process. Our results obtaining from running our Fano-BST-based algorithm on files of the Calgary corpus and the Canterbury corpus show that for small files, a large number of changes (shift operations) per encoding step are performed, whereas a few changes for each node are required when encoding larger files. This empirical analysis confirms our theoretical results demonstrated in Theorem 5.5 – the larger the number of encoded symbols, the more accurate the estimates of the input symbol probabilities, and hence less changes are performed in the Fano BST. All these results are also presented graphically.

Part III

Pattern Recognition: Overview and Research

Chapter 6

Pattern Recognition

The field of Pattern Recognition (PR) has become widely used in solving decision problems for quite a few decades. PR can be seen as the automation of a decision-making process based on some *a priori* information, and the information about the patterns to be recognized. This area primarily concerns "classifying" or "recognizing" a real life object that belongs to a certain class. Basically, PR comprises of two clearly defined phases: *learning* and *classification*. The former involves the analysis of the relevant *features* for the different classes. Excellent descriptions of the learning mechanisms used in PR can be found in [38, 47, 125, 146]. Classification concerns the decision process itself, i.e. the object is assigned to be belonging to one of the pre-determined classes participating in the classification.

Depending on the mechanisms, algorithms and structures used to represent the relevant information of the classes, PR itself can be considered to fall into two well defined approaches: Syntactic and Structural Pattern Recognition (SSPR), and Statistical Pattern Recognition (SPR). Basically, SSPR involves the use of *grammatical structures* to represent the objects being classified. The interested reader may refer to [146] for a detailed discussion of SSPR, which is out of the scope of this thesis. This thesis studies a few open problems in the field of SPR.

6.1 Statistical Pattern Recognition

6.1.1 Random Vectors

In the SPR paradigm, the features of the objects being recognized are represented as random vectors in the domain of integers or real numbers. A *random vector* is an ordered tuple $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_d]^T$, where d represents the dimension of the problem, and T denotes the transpose of the vector. Each component of the random vector, is a random variable, $\mathbf{x}_i$, which is defined as a function, $\mathbf{x}_i : \mathcal{S} \rightarrow \Re$, where $(\mathcal{S}, p)$ is a *probability space*, $\mathcal{S}$ is the *set of outcomes*, and p is a *probability function*. When $\mathbf{x}_i$ belongs to a discrete set, $\mathbf{x}_i$ is a *discrete random variable*. Conversely, $\mathbf{x}_i$ is a *continuous random variable*, if $\mathbf{x}_i$ belongs to a continuous set, e.g. the set of real numbers. In our study, we deal with continuous random variables, and more specifically, with the set of real numbers. Therefore, the features of an object are to be represented as a vector whose components are real numbers such as the *height*, *frequency of utterance*, *wavelength*, etc. If $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_d]^T$ is a random vector, a particular instantiation of $\mathbf{X}$ is denoted by $X = [x_1, \ldots, x_d]^T$.

Random variables are characterized by a *probability density function*, denoted by $p(x_i)$ and defined as a function, $p : \Re \rightarrow [0, 1]$. For random vectors, the probability density function is given by $p(X) = p(x_1, \ldots, x_d)$, where p is the function $p : \Re^d \rightarrow [0, 1]$, and represents the *joint probability density function* of $x_1, \ldots, x_d$. From this function, the marginal densities can also be calculated [47].

In most cases, the probability density function is represented by one or more mathematical functions. In this case, the distribution of a random variable is determined by its parameters. PR techniques using this model are known as *parametric* PR techniques. Conversely, *non-parametric* PR methods, which use *histograms* or the *k-nearest neighbor rule*, utilize other structures and techniques to represent the distribution of the object features [38, 47, 146].

One of the most widely used probability density functions is the *normal (or Gaussian) distribution*. Because of its numerous important properties, this distribution can be efficiently applied to many real problems [38, 47]. The probability density function of a normally distributed vector, X, is given by:

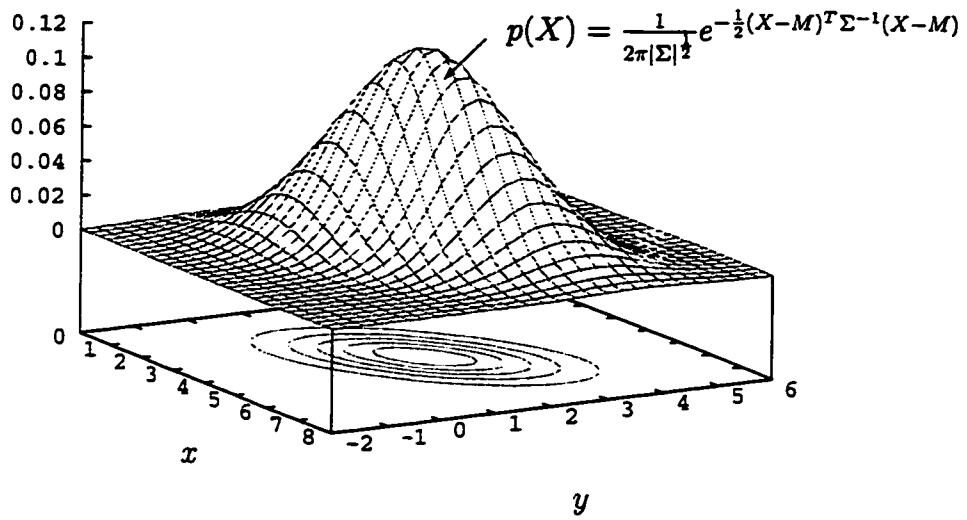


Figure 6.1: Plot of the probability density function for the two-dimensional normally distributed random vector of Example 6.1.

$$p(X) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-M)^T\Sigma^{-1}(X-M)}, \tag{6.1}$$

where M is the mean vector and Σ is the positive-definite, symmetric covariance matrix.

For our discussion, we let $\mathbf{X} \sim N(M, \Sigma)$ denote a normally distributed random vector, $\mathbf{X}$, with covariance matrix Σ and mean vector M.

Example 6.1. Consider a two-dimensional, normally distributed random vector, $\mathbf{X}$. Suppose that the distribution of $\mathbf{X}$ is characterized by the following parameters:

$$\Sigma = \begin{bmatrix} 2 & .2 \\ .2 & .4 \end{bmatrix} \text{ and } M = \begin{bmatrix} 4 \\ 2 \end{bmatrix}.$$

The shape of the probability density function is depicted in Figure 6.1. Observe that the curves composing the 3-D surface have the shape of density of the one dimensional

normal distribution since all the marginals are also normal. The surface can also be seen as composed of ellipses in the x-y plane. These ellipses are the projection of equally likely points onto the plane, or equivalently, the points with the same *Mahalanobis* distance [38, 162]. Observe too the contours of the plot in the plane $z = 0$. In this example, the coordinate axes of the ellipses are not in the direction of the coordinate axes of the system. As will be seen presently, whenever the covariance matrix is diagonal, both the coordinate axes are in the direction of the underlying reference frame, and are thus *isothetic*. □

6.1.2 Diagonalization

Diagonalization is the process of transforming a space by performing linear and whitening transformations [47]. As our linear classifier theory depends primarily on a preprocessing involving linear and whitening transformations, we present below a brief summary of this strategy.

Consider a normally distributed random vector $\mathbf{X} \sim N(M_X, \Sigma_X)$. The linear transformation consists of transforming the random vector $\mathbf{X}$ into another vector $\mathbf{Y}$ by performing $\mathbf{Y} = \Phi^T\mathbf{X}$, where Φ is a $d \times d$ matrix composed of the d eigenvectors of Σ. That is, Φ is given by $\Phi = [\phi_1 | \ldots | \phi_d]$. The distribution of $\mathbf{Y}$ in the transformed space is $\mathbf{Y} \sim N(\Phi^T M_X, \Phi^T \Sigma_X \Phi)$. Note that this is an orthonormal transformation, because it is well known that for any positive, definite matrix, Σ, its eigenvectors are orthogonal. In addition, the basis of the transformation, Φ, has to be normalized[1].

The whitening transformation is a non-orthonormal transformation consisting of $\mathbf{Z} = \Lambda^{-\frac{1}{2}}\mathbf{Y}$, where Λ is a diagonal matrix whose diagonal elements are the eigenvalues of Σ: $\lambda_1, \ldots, \lambda_d$. The distribution of $\mathbf{Z}$ in the transformed space results in

$$\mathbf{Z} \sim N(\Lambda^{-\frac{1}{2}}\Phi^T M_X, \Lambda^{-\frac{1}{2}}\Phi^T\Sigma_X\Phi\Lambda^{-\frac{1}{2}}) = N(\Lambda^{-\frac{1}{2}}\Phi^T M_X, I)\,, \tag{6.2}$$

since

[1] When each ϕ_i is normalized, $||\phi_i|| = 1$, where $||X||$ is the *norm* of X, and it is calculated as $\sqrt{\sum_{j=1}^{d} x_j^2}$.

$$\Lambda^{-\frac{1}{2}}\Phi^T\Sigma_X\Phi\Lambda^{-\frac{1}{2}} = \Lambda^{-\frac{1}{2}}\Lambda\Lambda^{-\frac{1}{2}} = I\,, \tag{6.3}$$

where I is the identity matrix.

6.1.2.1 Simultaneous Diagonalization

Suppose that we perform the orthonormal and the whitening transformations on two normally distributed random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$. The transformation of these two random vectors is called *simultaneous diagonalization.*

The first transformation consists of defining $\mathbf{Z}$ as $\mathbf{Z}_i = \Lambda_1^{-\frac{1}{2}}\Phi_1^T(\mathbf{Z}_i - M_1)$, $i = 1, 2$, where Λ_1 is the eigenvalue matrix of Σ_1, and Φ_1 is the eigenvector matrix of Σ_1. We subtract the mean vector of $\mathbf{X}_1$ so that the origin of our system is M_1. After this transformation, the resulting random vectors are $\mathbf{Z}_1 \sim N(\underline{0}, I)$ and $\mathbf{Z}_2 \sim N(\Lambda_1^{-\frac{1}{2}}\Phi_1^T[M_2 - M_1], \Lambda_1^{-\frac{1}{2}}\Phi_1^T\Sigma_2\Phi_1\Lambda_1^{-\frac{1}{2}})$.

In the second transformation, we set the axes of the coordinate system to be in the direction of the eigenvectors of $\Lambda_1^{-\frac{1}{2}}\Phi_1^T\Sigma_2\Phi_1\Lambda_1^{-\frac{1}{2}} = \Sigma_2'$. The process consists of the transformation $\mathbf{V}_i = \Phi_2^T\mathbf{Z}_i$, $i = 1, 2$, where Φ_2 is the eigenvector matrix of Σ_2'. After the transformation, the resulting random vectors are $\mathbf{V}_1 \sim N(\underline{0}, I)$ and $\mathbf{V}_2 \sim N(\Phi_2^T\Lambda_1^{-\frac{1}{2}}\Phi_1^T[M_2 - M_1], \Phi_2^T\Sigma_2'\Phi_2)$. After this transformation, the covariance matrix of $\mathbf{V}_2$ is Λ_2, a diagonal matrix whose diagonal elements are the eigenvalues of Σ_2'.

One of the implications of the simultaneous diagonalization is that $\mathbf{Z}_1$ is in the origin (i.e. its mean vector is $\underline{0}$), and its covariance matrix is the identity. This means that any orthonormal transformation, such as $\mathbf{V}_i = \Phi_2^T\mathbf{Z}_i$, does not change the distribution of $\mathbf{Z}_1$. In addition, both covariance matrices are diagonal, and so their eigenvectors are parallel to the axes of the coordinate system. Additionally, the random variables, which compose the random vectors, are uncorrelated and independent.

Example 6.2. Consider two normally distributed random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$, whose parameters are given by:

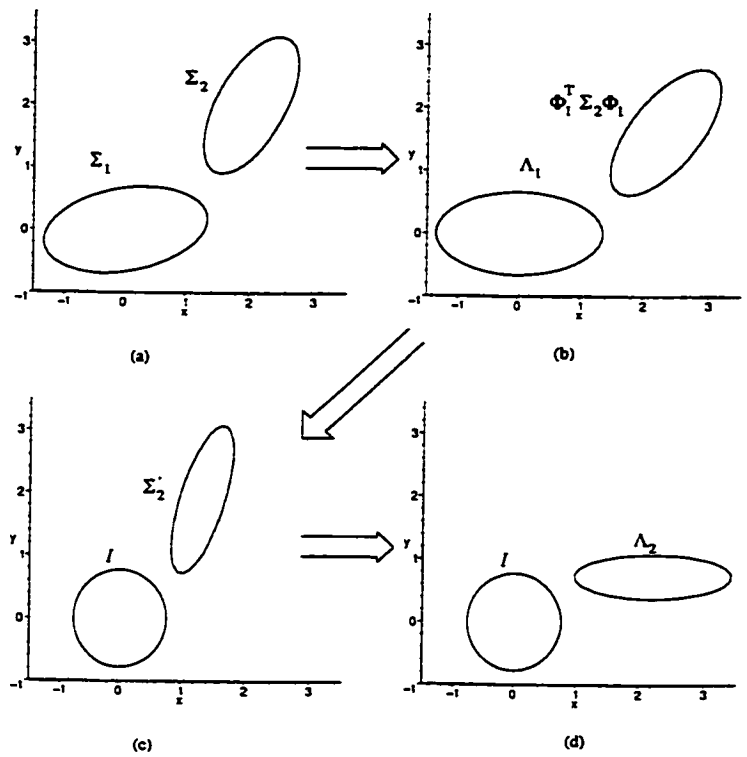


Figure 6.2: Simultaneous diagonalization: orthonormal and whitening transformations.

$$M_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, M_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 3 & .4 \\ .4 & .8 \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} .9 & .8 \\ .8 & .2 \end{bmatrix}.$$

The simultaneous diagonalization procedure is depicted in Figure 6.2. The distributions of the classes are plotted by considering points with the same probability, i.e. those vectors X_1 and X_2 whose probabilities are $p(X_1) \approx 0.07872851279$ and $p(X_2) \approx 0.1094719344$ respectively. For both X_1 and X_2, we plot the points for which the Mahalanobis distance is 0.6. The original distributions (for $\mathbf{X}_1$ and $\mathbf{X}_2$) are depicted in Figure 6.2(a). Figure 6.2(b) represents the corresponding normal random vectors after the orthonormal transformation $\mathbf{Y}_i = \Phi_1^T(\mathbf{X}_i - M_1)$ with the axes of the system in the direction of the eigenvectors of Σ_1. The whitening transformation $\mathbf{Z}_i = \Lambda_1^{-\frac{1}{2}}\mathbf{Y}_i$ is depicted in Figure 6.2(c); the new value of Σ_1 is the identity matrix drawn as a circle. The resulting normal random vectors obtained after the orthonormal transformation $\mathbf{V}_i = \Phi_2^T\mathbf{Z}_i$ are depicted in Figure 6.2(d). Observe that the axes of the

system are in the direction of the eigenvectors of Σ_2'.

Finally, the parameters of the new random vectors in the transformed space, $\mathbf{V}_1$ and $\mathbf{V}_2$, are:

$$M_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, M_2 \approx \begin{bmatrix} 2.1975 \\ .7267 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } \Sigma_2 \approx \begin{bmatrix} 2.5078 & 0 \\ 0 & .2065 \end{bmatrix}.$$

□

6.2 Pattern Classification

6.2.1 Bayes Decision Theory

In general, an SPR problem deals with c classes, $\omega_1, \ldots, \omega_c$, where the *a priori* probability of ω_i is $P(\omega_i)$. Consider the problem of classifying an object whose features are given by the vector $X = [x_1, \ldots, x_d]^T$. The aim of Bayesian classification is to decide and choose the class that maximizes the *a posteriori* probability [38, 54, 55], given by:

$$p(\omega_i|X) = \frac{p(X|\omega_i)P(\omega_i)}{p(X)}. \tag{6.4}$$

In this thesis, we assume that we have two classes, ω_1 and ω_2, with *a priori* probability, $P(\omega_1)$ and $P(\omega_2)$. From Equation (6.4), we can write the general inequality specifying the Bayesian classification between two classes as follows:

$$p(X|\omega_1)P(\omega_1) \underset{\omega_1}{\overset{\omega_2}{\lessgtr}} p(X|\omega_2)P(\omega_2), \tag{6.5}$$

where the upper inequality ('<' in this case) implies that ω_2 must be chosen, and the other inequality ('>' in this case) means that ω_1 must be chosen.

Equality in (6.5) represents the *discriminant* function. Assuming that ω_1 and ω_2 are represented by normal random vectors with covariance matrices Σ_1 and Σ_2, and

mean vectors M_1 and M_2, respectively, the discriminant function is given by:

$$\left[\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_1|^{\frac{1}{2}}}e^{-\frac{1}{2}(X-M_1)^T\Sigma_1^{-1}(X-M_1)}\right]P(\omega_1) = \left[\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_2|^{\frac{1}{2}}}e^{-\frac{1}{2}(X-M_2)^T\Sigma_2^{-1}(X-M_2)}\right]P(\omega_2)\,. \tag{6.6}$$

Without loss of generality, we assume that ω_1 and ω_2 have the same *a priori* probability, 0.5. Taking the logarithm of both sides of (6.6), and doing some algebraic manipulations, we have:

$$\log\frac{|\Sigma_2|}{|\Sigma_1|} - (X-M_1)^T\Sigma_1^{-1}(X-M_1) + (X-M_2)^T\Sigma_2^{-1}(X-M_2) = 0\,. \tag{6.7}$$

After some algebraic operations, and using the fact that Σ_1 and Σ_2 are symmetric, (6.7) can be written as follows:

$$X^T(\Sigma_2^{-1}-\Sigma_1^{-1})X + 2(M_1^T\Sigma_1^{-1} - M_2^T\Sigma_2^{-1})X - M_1^T\Sigma_1^{-1}M_1 + M_2^T\Sigma_2^{-1}M_2 + \log\frac{|\Sigma_2|}{|\Sigma_1|} = 0\,. \tag{6.8}$$

Equation (6.8) is a general equation of second degree in its matrix form. Whenever Σ_1 and Σ_2 are arbitrary, the classifier results in a general equation of second degree, and the decision boundary is a *hyperquadric* which has the form of a hyperparaboloid, a hyperellipsoid, a hypersphere, a hyperboloid, a hyperplane, or a pair of hyperplanes.

We shall now catalogue the various PR problems that we have solved.

6.3 Pairwise Linear Classifiers

6.3.1 Single Hyperplane Classifiers

As discussed in Section 6.2.1, the discriminant function for the optimal Bayesian classifier can be either linear or quadratic. When the classes are normally distributed, it is not always possible to find an optimal linear classifier. In all the known results in this field, determining a linear function to achieve Bayes classification for normally distributed class-conditional distributions, has only been reported when the covariance matrices are equal [38, 47, 78, 125, 146, 162].

When $\Sigma_1 = \Sigma_2 = \Sigma$, it is clear that (6.8) is reduced to a linear equation of the form:

$$B^T X + C = 0\,, \tag{6.9}$$

where $B^T = 2(M_1 - M_2)^T \Sigma^{-1}$ is a $1 \times d$ matrix (a d-dimensional vector), and $C = M_1^T \Sigma^{-1} M_1 + M_2^T \Sigma^{-1} M_2$ is a real number. Note that $\log \frac{|\Sigma_2|}{|\Sigma_1|} = 0$, since $\Sigma_1 = \Sigma_2 = \Sigma$.

In this case, the discriminant function is *linear* (a single hyperplane) and *optimal.*

Example 6.3. Consider two normally distributed random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$, representing two classes, ω_1 and ω_2, respectively. Suppose that the parameters of $\mathbf{X}_1$ and $\mathbf{X}_2$ are the following:

$$M_1 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, M_2 \approx \begin{bmatrix} 4 \\ 2 \end{bmatrix}, \Sigma_1 = \Sigma_2 = \begin{bmatrix} 4.5 & 0.75 \\ 0.75 & 2 \end{bmatrix}.$$

Using (6.9) we obtain the optimal linear classifier, which is given by the following equation:

$$\begin{bmatrix} -1.3037 \\ 2.48889 \end{bmatrix} X - 3.55556\,.$$

This scenario is depicted in Figure 6.3. Observe that both classes have the same shape, but different means. The linear classifier is given by the implicit equation (shown in the figure):

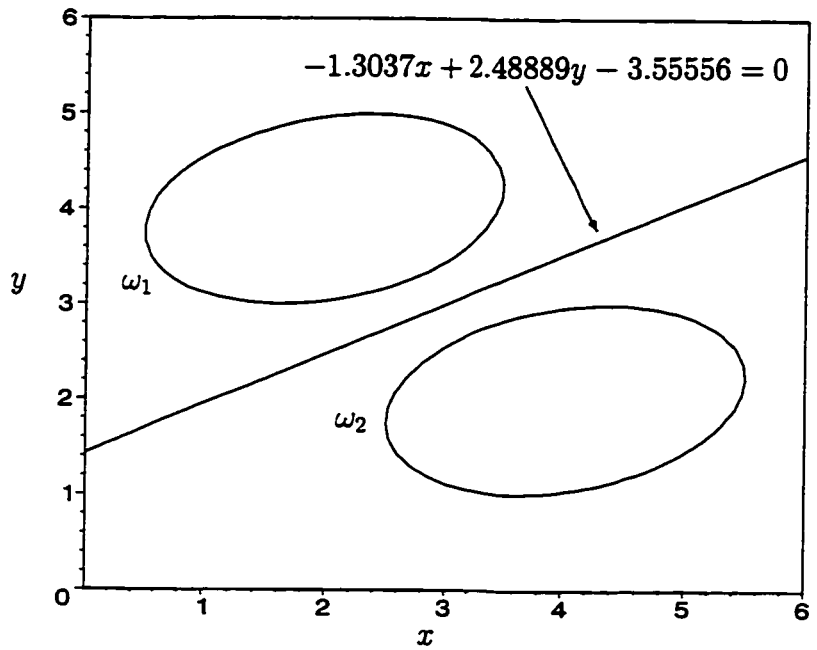
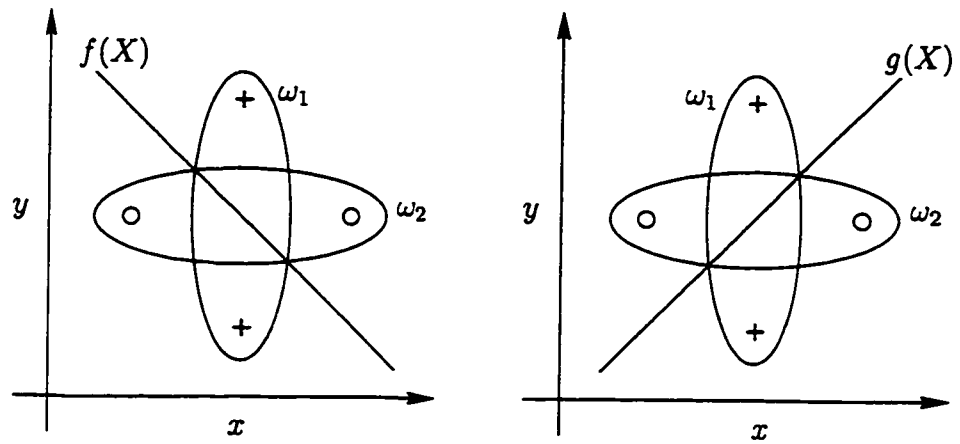


Figure 6.3: Optimal linear classifier between two normally distributed classes, ω_1 and ω_2, given in Example 6.3.

$$-1.3037x + 2.48889y - 3.55556 = 0\,,$$

where $X = \begin{bmatrix} x \\ y \end{bmatrix}$. □

As opposed to optimal linear classifiers, many attempts have been made to yield linear classifiers, including *Fisher's approach* [38, 89, 146], the *perceptron algorithm* (the basis of the back propagation *neural networks* learning algorithms) [87, 101, 122, 123], *piecewise recognition models* [119], *random search optimization* [121], *removal classification structures* [5], *adaptive linear dimensionality reduction* [88] (which outperforms Fisher's classifier for some data sets), and *linear constrained distance-based classifier analysis* [36] (an improvement to Fisher's approach designed for hyperspectral image classification). All of these approaches suffer from the lack of optimality, and thus although they find linear discriminant functions, the classifier is not optimal.



(a) A single line classifier.

(b) Another single line classifier.

Figure 6.4: Two overlapping classes, ω_1 and ω_2, with equal means, and possible ways of classification using straight lines.

6.3.2 Optimal Pairwise Linear Classifiers

In this thesis, we show that there are other cases for normal distributions and non-equal covariance matrices in which the discriminant function is linear and the classifier is *optimal*. One of these cases is when considering two overlapping classes with equal means, as depicted in Figure 6.4. But as opposed to all the previously studied linear classifiers, the new techniques introduced here yield *pairwise* linear classifiers, which emerge as degenerate cases of the general quadratic classifier.

In 1957, a famous mathematician, M. Minsky, showed that it is not possible to find a single linear classifier for the simple case of Figure 6.4 in which the features of one class are the exclusive-OR of a 2-bit binary vector, and the features of the second class are the negated features. In the figure, one of the classes, ω_1, is represented by the symbol '+', and the other one, ω_2, by the symbol 'o'. Observe that neither of the two possible linear classifiers, $f(x)$ or $g(x)$ (the cases depicted in Figure 6.4(a) and (b) respectively) can correctly classify samples of these classes. This paradox, called *Minsky's paradox* [96], demonstrated that a single perceptron could not correctly classify in this simple scenario.

The above-mentioned problem is also called the *two-bit parity* or the *exclusive-OR* problem [38]. Using neural networks, it can be shown that a network with *at least* three layers, and which involves a non-linear classifier can achieve this classification.

As opposed to this, we show that it is possible to find an optimal linear discriminant function, specified as a pair of hyperplanes, which is a particular case of the quadratic discriminant function. These classifiers have some advantages over the traditional linear discriminant approaches, such as Fisher's approach, the perceptron learning for neural networks, and other ones, because the classifier that we obtain is both *linear* and *optimal*.

In Chapter 7, we center our discussion on the case in which the classifier is a pair of straight lines for two dimensional normally distributed random vectors. In Chapter 8, we continue to study and analyze multi-dimensional optimal pairwise linear classifiers for normal distributions. The initial results for the optimal pairwise linear classifiers for the two dimensional case have been published in [137, 138]. The corresponding formal theory for the multi-dimensional optimal pairwise linear classifiers has been published in [134, 139].

Finally, we conclude this section by observing that, to the best of our knowledge, the results presented here are pioneering. We are unaware of any work that has been done in SPR, which investigates the design and use of optimal pairwise linear classifiers.

6.4 Heuristic Functions: Accuracy and Optimality

Computer scientists face many open, unsolved problems. Many of these problems have been proved to be NP-hard, and hence it is unlikely that polynomial-time algorithms that solve these problems exist unless P = NP. A typical NP-hard problem is an optimization problem which can also be formulated as a decision problem [49]. In this research work we are concerned with one such problems, namely that of using *heuristics* to solve optimization problems.

Any arbitrary optimization problem[2] is typically defined in terms of instances which are drawn from a (finite) set, $\mathcal{X}$, an objective function, and some feasibility functions.

[2]Every optimization problem can also be formulated as a decision problem [49].

The aim is to find an (and hopefully, the unique) instance of $\mathcal{X}$, which leads to the maximum (or the minimum) value of the objective function subject to the feasibility constraints.

A *heuristic algorithm* is an algorithm that attempts to find a certain instance of $\mathcal{X}$ that maximizes f (or the *profit*) by iteratively invoking a *heuristic* function. The instance that maximizes f will be the *optimal solution*[3] to the optimization problem. A *heuristic* is a method that performs one or more modifications to a given solution or instance, in order to obtain a different solution which is either superior, or which leads to a superior solution. The heuristic, in turn, invokes a heuristic function, which estimates (or measures) the cost of the solution at the particular state in the search process.

Many heuristic algorithms and heuristic functions have been reported in the literature, where the former include the *alpha-beta search* [86], *backtracking*, *hill-climbing* [77], *simulated annealing* [1], *genetic algorithms* [95], *tabu search* [52], *learning automata* [102], etc. The issue of how heuristic functions are used in such heuristic algorithms in searching, game playing, etc., can be found in [104, 124] and is, indeed, an enormous field of study in itself. This question is not addressed here.

A typical heuristic algorithm is an algorithm that utilizes one or more heuristic functions to estimate the cost of a solution. To the best of our knowledge, the formal relationship between heuristic function accuracy and solution optimality has been an open problem. Indeed, it has been the *tacit unproven conjecture which has served as the basis for designing various searching algorithms such as the A^* algorithm, and its variants* [104, 124, 128, 172].

This is the second problem that we will solve using PR techniques in this section of the thesis. Our solution can be summarized as follows: Given two heuristic functions, H_1 and H_2, used by a heuristic algorithm in finding a solution to a particular problem, if the accuracy in obtaining the *optimal* solution by using H_1 is greater than that of using H_2, then H_1 has a higher probability of leading to the optimal solution than H_2. In contrast to the previous research, in which this question demanded a rigorous yes/no

[3]We use the term "solution" to refer to an element $x \in \mathcal{X}$, and the term "profit" to refer to the value of $f(x)$. In minimization problems, $f(.)$ will be a cost function.

answer, our solution states that we can provide a a *stochastically* positive/negative answer.

In our model, we first represent the distribution of the accuracy of the solutions by doubly exponentially distributed random variables. We present an analytical, rigorous proof of this conjecture for this distribution. We then proceed to extend the analysis for the Gaussian distribution, for which we submit a computational analysis, since a formal analytic proof is infeasible due to the constraints involved in integrating the normal probability density function.

The problems in which this model can be applied, can be found in a wide range of applications spanning the whole spectrum of *artificial intelligence*, and include *game playing* and *game theory*, *graph theory*, *database query optimization*, *networking*, *computational geometry*, *number theoretic problems*, *parallel processing*, etc. The results presented in Chapter 9 are general enough to be applicable to any heuristic algorithm (which could utilize any heuristic function), and for any particular problem. However, in this research work, we have empirically concentrated on only the database query optimization problem, for which we present empirical results that show the validity of our results for a few well-known histogram estimation methods tested on randomly generated databases and randomly generated queries.

6.5 Conclusions

In this chapter, we first presented a brief introduction to statistical pattern recognition. We discussed the theory of random vectors for normal distributions. We briefly surveyed the formal procedure for the diagonalization process, which involves an orthonormal transformation and a whitening transformation. The process of simultaneous diagonalization is quite useful in our study, as we will see later in Chapter 7.

We also briefly iterated the formal theory for pattern classification using the Bayes decision theory and normal distributions.

The body of the chapter discusses the various problems which we will be investigating in this thesis.

The first problem that we study in the area of PR relates to optimal Bayesian

linear classifiers. When dealing with two normally distributed classes whose covariance matrices are Σ_1 and Σ_2, the optimal classifier is known to be linear only when the covariance matrices are equal. As opposed to this, we shall derive the necessary and sufficient conditions for this classifier to be of the form of a *pair of hyperplanes*. This scenario is the generalized version of the so-called Minsky's paradox, which has remained unsolved since 1957. The formal analysis for this problem has been given in Chapter 7 for the two dimensional case, and in Chapter 8 for the multi-dimensional case. In every case our results have been verified by empirical simulations - including those involving the resolution of Minsky's paradox.

Finally, we mentioned the problem of establishing the relation between accuracy and optimality when using heuristic functions for solving various optimization problems. We have defined this relationship using Pattern Classification Techniques. The formal analysis for this is included in Chapter 9, and is verified by numerous empirical results.

Chapter 7

2-Dimensional Optimal Pairwise Linear Classifiers

As discussed in Section 6.2.1, the discriminant function for the optimal Bayesian classifier can be either linear or quadratic. When the classes are normally distributed, it is not always possible to find an optimal linear classifier. In all the known results in this field, determining a linear function to achieve Bayes classification for normally distributed class-conditional distributions, has only been reported when the covariance matrices are equal [37, 47, 78, 125, 146].

In this chapter, we show that there are other cases for normal distributions and non-equal covariance matrices in which the discriminant function is linear and the classifier is *optimal*. One of these cases is when we have two overlapping classes with equal means. But as opposed to all the previously studied linear classifiers, the new techniques introduced here yield *pairwise* linear classifiers, which emerge as degenerate cases of the general quadratic classifier.

Minsky showed that it is not possible to find a single linear classifier for the simple case in which the features of one class are the Exclusive-OR of a 2-bit binary vector and the features of the second class are the negated features. Minsky's paradox [96] demonstrated that a single perceptron could not correctly classify in this simple scenario. This problem is also known as the *two-bit parity* or the *exclusive-OR* problem, and can only be solved with a neural network of at least three layers [38].

As opposed to this, we show that it is possible to find two optimal linear discriminant functions, given as a pair of straight lines, (which is a particular case of the quadratic discriminant function) even for the scenario when the classes are not represented as points but distributions with well-defined covariance matrices. These classifiers have some advantages over the traditional linear discriminant approaches, such as Fisher's, perceptron learning, and other ones, because the classifier that we obtain is both linear and optimal.

For the rest of this chapter we focus our discussion on the case in which the classifier is a pair of straight lines for two dimensional random vectors. These results have been published in [136, 138], and will be later generalized for the multi-dimensional case in Chapter 8.

7.1 Pairwise Linear Classifiers : Cases

7.1.1 Diagonalized Classes

As discussed in Section 6.1.2, given any arbitrary normal random vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, whose covariance matrices are Σ_1 and Σ_2, we can perform simultaneous diagonalization to obtain normal random vectors, $\mathbf{V}_1$ and $\mathbf{V}_2$, whose covariance matrices are diagonal, namely I and Λ, respectively. After simultaneous diagonalization, the mean vectors and covariance matrices have the form:

$$M_1 = \begin{bmatrix} p \\ q \end{bmatrix}, M_2 = \begin{bmatrix} r \\ s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}. \tag{7.1}$$

Since we will be, for the present, consistently dealing with two-dimensional vectors we shall assume that the feature vector has the form $X = \begin{bmatrix} x \\ y \end{bmatrix}$.

We will now present a linear transformation that will later prove useful in simplifying complex expressions. The transformation is stated more formally in Theorem 7.1.

Theorem 7.1. Let $\mathbf{X}_1 \sim N(M_{1_X}, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_{2_X}, \Sigma_2)$ be two normal random vectors with parameters as in (7.1). Vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ can be transformed into $\mathbf{Z}_1 \sim N(M_{1_Z}, \Sigma_1)$ and $\mathbf{Z}_2 \sim N(M_{2_Z}, \Sigma_2)$, where

$$M_{1_Z} = \begin{bmatrix} t \\ u \end{bmatrix}, \text{ and } M_{2_Z} = \begin{bmatrix} -t \\ -u \end{bmatrix}. \tag{7.2}$$

Proof. Suppose that we perform the following linear transformation:

$$\mathbf{Z}_1 = \mathbf{X}_1 - \left\{ M_{2_X} + \frac{M_{1_X} - M_{2_X}}{2} \right\}, \text{ and } \mathbf{Z}_2 = \mathbf{X}_2 - \left\{ M_{2_X} + \frac{M_{1_X} - M_{2_X}}{2} \right\}.$$

As a result of this transformation, Σ_1 and Σ_2 will remain unchanged. The new mean vector of $\mathbf{Z}_1$ results in

$$M_{1_Z} = M_{1_X} - \left\{ M_{2_X} + \frac{M_{1_X} - M_{2_X}}{2} \right\} = \begin{bmatrix} p \\ q \end{bmatrix} - \begin{bmatrix} r \\ s \end{bmatrix} - \frac{\begin{bmatrix} p \\ q \end{bmatrix} - \begin{bmatrix} r \\ s \end{bmatrix}}{2} = \begin{bmatrix} \frac{p-r}{2} \\ \frac{q-s}{2} \end{bmatrix}.$$

In the same way, the new mean vector of $\mathbf{Z}_2$ results in

$$M_{2_Z} = M_{2_X} - \left\{ M_{2_X} + \frac{M_{1_X} - M_{2_X}}{2} \right\} = \begin{bmatrix} r \\ s \end{bmatrix} - \begin{bmatrix} r \\ s \end{bmatrix} - \frac{\begin{bmatrix} p \\ q \end{bmatrix} - \begin{bmatrix} r \\ s \end{bmatrix}}{2} = \begin{bmatrix} -\frac{p-r}{2} \\ -\frac{q-s}{2} \end{bmatrix}.$$

Therefore, the new random vectors are $\mathbf{Z}_1 \sim N(M_{1_Z}, \Sigma_1)$ and $\mathbf{Z}_2 \sim N(M_{2_Z}, \Sigma_2)$, where

$$M_{1_Z} = \begin{bmatrix} \frac{p-r}{2} \\ \frac{q-s}{2} \end{bmatrix}, \text{ and } M_{2_Z} = \begin{bmatrix} -\frac{p-r}{2} \\ -\frac{q-s}{2} \end{bmatrix}.$$

□

For two normally distributed random vectors with means of the form of (7.2) and diagonal covariance matrices, we are now interested in finding the necessary and sufficient conditions for yielding the optimal pairwise linear classifier.

Theorem 7.2. Let $X_1 \sim N(M_1, \Sigma_1)$ and $X_2 \sim N(M_2, \Sigma_2)$ be two normally distributed random vectors, such that:

$$M_1 = \begin{bmatrix} r \\ s \end{bmatrix}, M_2 = \begin{bmatrix} -r \\ -s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}. \tag{7.3}$$

The optimal classifier obtained by Bayes classification is a pair of straight lines *if and only if* there exist positive real numbers, a and b, such that:

$$a(1-b)r^2 - \frac{1}{4}(ab - a - b + 1)\log ab = (a-1)bs^2. \tag{7.4}$$

Proof. The discriminant function given in Equation (6.7) is now:

$$\log|\Sigma_2| - \begin{bmatrix} x-r \\ y-s \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x-r \\ y-s \end{bmatrix} + \begin{bmatrix} x+r \\ y+s \end{bmatrix}^T \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x+r \\ y+s \end{bmatrix} = 0, \tag{7.5}$$

After performing rather lengthy matrix operations, Equation (7.5) results in:

$$a(x+r)^2 + b(y+s)^2 - (x-r)^2 - (y-s)^2 - \log ab = 0. \tag{7.6}$$

Expanding the quadratic terms and grouping by x and y, we have:

$$(a-1)x^2+(b-1)y^2+2(a+1)rx+2(b+1)sy+ar^2+bs^2-r^2-s^2-\log ab=0\,. \tag{7.7}$$

Equation (7.7) is a general equation of second degree, and can represent either a circle, an ellipse, a parabola, a hyperbola, or a pair of straight lines. Indeed, we are interested in the latter. Equation (7.7) represents a pair of straight lines *if and only if* the following condition is satisfied [17]:

$$(a-1)(b-1)(ar^2+bs^2-r^2-s^2-\log ab)-(a-1)[(b+1)s]^2-(b-1)[(a+1)r]^2=0 \tag{7.8}$$

Equation (7.8) is equivalent to:

$$-4abr^2-4abs^2+4ar^2+4bs^2-(ab-a-b+1)\log ab=0 \tag{7.9}$$

Multiplying both sides of (7.9) by $\frac{1}{4}$, and grouping some terms, we get:

$$a(1-b)r^2-\frac{1}{4}(ab-a-b+1)\log ab=(a-1)bs^2\,. \tag{7.10}$$

We have proved[1] that (7.8) $\equiv$ (7.9) $\equiv$ (7.10), all of which are *iff* assertions. The result follows. □

Equation (7.10) is the necessary and sufficient condition that real numbers $a > 0, b > 0, r$, and s, must satisfy in order to yield the optimal linear classifier between two classes represented by normal random vectors with parameters of the form given in (7.3).

[1]Easier methods of analysis seem to be possible if (7.10) is written as $a(1-b)r^2+(1-a)bs^2=\frac{1}{4}(1-a)(1-b)\log ab$. We are grateful to Dr. Kaizhong Zhang for pointing out this observation during the final defense.

Consider the following: Given positive real numbers $a > 0$ and $b > 0$, we would like to find real numbers, r and s, that satisfy (7.10). When neither a nor b equals unity, we have four possible cases:

$$\left.\begin{array}{ll} (i) & 0 < a < 1,\ b > 1, \\ (ii) & a > 1,\ 0 < b < 1, \\ (iii) & 0 < a < 1,\ 0 < b < 1, \text{ and} \\ (iv) & a > 1,\ b > 1\,. \end{array}\right\} \tag{7.11}$$

The cases for which it is possible to find real numbers, r and s, satisfying Equation (7.10) are stated and proved in Theorem 7.3 below. These constitutes the necessary and sufficient conditions as will be explained presently.

Theorem 7.3. Let $X_1 \sim N(M_1, \Sigma_1)$ and $X_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors, such that

$$M_1 = \begin{bmatrix} r \\ s \end{bmatrix}, M_2 = \begin{bmatrix} -r \\ -s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix} \tag{7.12}$$

For any positive real numbers a and b ($a \neq 1$, $b \neq 1$), there exist real numbers r and s permitting pairwise linear classification *if and only if* $a > 1$ and $0 < b < 1$, OR $0 < a < 1$ and $b > 1$.

Proof. IF: We have to prove that if a and b are as in Cases(i) and (ii) of (7.11), there exist real numbers, r and s, permitting pairwise linear classification.

Consider the non-quadratic term of Equation (7.10), $l = \frac{1}{4}(ab - a - b + 1)\log ab$.

(*i*) $0 < a < 1$ and $b > 1$.

We know that

$$0 < a < 1 \Rightarrow -1 < a - 1 < 0 \Rightarrow (a-1)b < 0 \text{ since } b > 1, \text{ and}$$

$$b > 1 \Rightarrow -b < -1 \Rightarrow 1 - b < 0 \Rightarrow a(1-b) < 0 \text{ because } a > 0.$$

Since $s^2 \geq 0$, (7.10) can be written as follows:

$$s^2 = \frac{a(1-b)r^2 - l}{(a-1)b} \geq 0. \tag{7.13}$$

We are now required to find a real number r such that (7.13) is satisfied.

Since $(a-1)b < 0$, we must find r such that the following inequality is satisfied:

$$a(1-b)r^2 \leq l \;\equiv\; r^2 \geq \frac{l}{a(1-b)} \tag{7.14}$$

Utilizing the fact that $r^2 \geq 0$, we can argue that we can choose r such that r^2 is sufficiently large so as to satisfy the inequality given in (7.14).

(*ii*) $a > 1$ and $0 < b < 1$.

We know that

$0 < b < 1 \Rightarrow 0 > -b > -1 \Rightarrow 1 - b > 0 \Rightarrow a(1-b) > 0$ since $a > 1$, and

$a > 1 \Rightarrow a - 1 > 0 \Rightarrow b(a-1) > 0$ because $b > 0$.

Since $s^2 \geq 0$, (7.10) can be written as follows:

$$s^2 = \frac{a(1-b)r^2 - l}{(a-1)b} \geq 0. \tag{7.15}$$

We are now required to find a real number r such that (7.15) is satisfied.

Since $(a-1)b > 0$, we must find r such that the following inequality is satisfied:

$$r^2 \geq \frac{l}{a(1-b)}. \tag{7.16}$$

But we know that r^2 is always positive, and so we can choose r such that r^2 is sufficiently large so as to satisfy (7.16).

Cases I and II prove the sufficiency conditions.

ONLY IF: We are to prove that if the classifier is pairwise linear, a and b fall in Case either (i) or (ii). We prove this by showing that the negation of the implication is true, or equivalently if a and b do not fall in Cases (i) and (ii) (i.e. they fall in cases (iii) or (iv)), it is not possible to find real numbers r and s permitting pairwise linear classification.

(*iii*) $0 < a < 1$ and $0 < b < 1$.

In this case, we know that

$$0 < a < 1 \Rightarrow (a-1)b < 0 \text{ and } 0 < b < 1 \Rightarrow a(1-b) > 0.$$

Since $r^2 \geq 0$ and $s^2 \geq 0$, $(a-1)bs^2 \leq 0$ and $a(1-b)r^2 \geq 0$.

We shall show that this implies that if $l < 0$, it is not possible to find real numbers r and s.

Suppose that it were possible to find real numbers, r, s, and $l < 0$. This implies that

$$(ab - a - b + 1)\log ab > 0, \text{and}$$

$$ab - a - b + 1 < 0 \tag{7.17}$$

since $\log ab < 0$.

By our hypothesis, we know that

$$0 < a < 1 \text{ and } 0 < b < 1 \Rightarrow 0 < a + b - ab < 1 \Rightarrow ab - a - b + 1 > 0,$$

which contradicts (7.17). Therefore, there are no real numbers r and s that satisfy Equation (7.10) for $0 < a < 1$ and $0 < b < 1$.

(*iv*) $a > 1$ and $b > 1$.

In this case, we know that

$$a > 1 \Rightarrow (a-1)b > 0 \text{ and } b > 1 \Rightarrow a(1-b) < 0.$$

Since $r^2 \geq 0$ and $s^2 \geq 0$, then $(a-1)br^2 \geq 0$ and $a(1-b)s^2 \leq 0$.

We again argue that this implies that if $l < 0$, it is not possible to find real number r and s.

Suppose that we are able to find real numbers, r, s, and $l < 0$. This implies that

$$(ab - a - b + 1)\log ab < 0, \text{and}$$

$$ab - a - b + 1 > 0, \tag{7.18}$$

since $\log ab > 0$.

By our hypothesis, we know that

$$a > 1 \text{ and } b > 1 \Rightarrow a + b - ab > 1 \Rightarrow ab - a - b + 1 < 0,$$

which contradicts (7.18). Therefore, there are no real numbers r and s that satisfy Equation (7.10) for $a > 1$ and $b > 1$.

Hence the necessary conditions.

The theorem is thus proved. □

In the above theorem, we considered the cases only when both $a \neq 1$ and $b \neq 1$. The case when either a or b, or both, is unity is given below.

Theorem 7.4. Let $X_1 \sim N(M_1, \Sigma_1)$ and $X_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors with parameters of the form (7.12). When either a or b can have the value of unity, the optimal Bayesian classifier is a pair of straight lines *if and only if*:

(a) $a = 1$, $b \neq 1$, and $r = 0$,

(b) $a \neq 1$, $b = 1$, and $s = 0$, or

(c) $a = 1$ and $b = 1$.

Proof. The proof of the above cases can be derived by substituting these conditions into the relevant expressions of Theorem 7.3. In Case (iii), we have the situation in which $\Sigma_1 = \Sigma_2 = I$, which is the only scenario known to have a linear discriminant function as the classifier (i.e. a *single* straight line[2]) [47]. □

7.1.2 Different Covariances and Different Means

Consider two normal random vectors of dimension $d = 2$. Using the diagonalization process discussed in Section 6.1.2, any covariance matrices and mean vectors can be converted into the form of (7.1). Starting with normal random vectors and these parameters, we are interested in analyzing linear classifiers for a more particular case.

Theorem 7.5. Let X_1 and X_2 be two normal random vectors with covariance matrices and mean vectors as in (7.1). It is possible to transform X_1, X_2 into $Z_1 = A^T X_1$, $Z_2 = A^T X_2$, respectively, where $A = \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix}$, and the new covariance matrices and mean vectors have the form:

$$M_{1_Z} = \begin{bmatrix} p' \\ q' \end{bmatrix}, M_{2_Z} = \begin{bmatrix} r' \\ s' \end{bmatrix}, \Sigma_{1_Z} = \begin{bmatrix} a' & 0 \\ 0 & b' \end{bmatrix}, \text{ and } \Sigma_{2_Z} = \begin{bmatrix} b' & 0 \\ 0 & a' \end{bmatrix} \tag{7.19}$$

[2]Note that a single straight line means that the quadratic polynomial of (7.7) has a single root, or in the richer context of this paper, it has *two* coincident roots.

if and only if $b = a^{-1}$.

Proof. Consider a matrix $A = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$ where α and β are any real numbers. The covariance matrices in the transformed space are:

$$\Sigma_{1_Z} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} = \begin{bmatrix} \alpha^2 & 0 \\ 0 & \beta^2 \end{bmatrix}, \qquad \text{and}$$

$$\Sigma_{2_Z} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}^T \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} = \begin{bmatrix} a^{-1}\alpha^2 & 0 \\ 0 & b^{-1}\beta^2 \end{bmatrix}.$$

In order to obtain the form of (7.19), we need that $\alpha^2 = b^{-1}\beta^2$ and $a^{-1}\alpha^2 = \beta^2$. Substituting α^2 into the second expression, we have $a^{-1}b^{-1}\beta^2 = \beta^2 \Rightarrow b = a^{-1}$. Therefore, the covariance matrix of X_2 must be of the form $\Sigma_{2_X} = \begin{bmatrix} a^{-1} & 0 \\ 0 & a \end{bmatrix}$.

By choosing $A = \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix}$, the covariance matrices in the transformed space are:

$$\Sigma_{1_Z} = \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix}^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix} = \begin{bmatrix} a^{\frac{1}{2}} & 0 \\ 0 & a^{-\frac{1}{2}} \end{bmatrix}, \text{ and}$$

$$\Sigma_{2_Z} = \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix}^T \begin{bmatrix} a^{-1} & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} a^{\frac{1}{4}} & 0 \\ 0 & a^{-\frac{1}{4}} \end{bmatrix} = \begin{bmatrix} a^{-\frac{1}{2}} & 0 \\ 0 & a^{\frac{1}{2}} \end{bmatrix}.$$

The mean vectors in the transformed space are $M_{1_Z} = \begin{bmatrix} a^{\frac{1}{4}}p \\ a^{-\frac{1}{4}}q \end{bmatrix}$ and $M_{2_Z} = \begin{bmatrix} a^{\frac{1}{4}}r \\ a^{-\frac{1}{4}}s \end{bmatrix}$.

The "only if" part follows by traversing the algebraic steps in a reverse manner. □

We now state the conditions necessary to obtain a pair of straight lines when the parameters of the distributions have the form of (7.19).

Theorem 7.6. Let $X_1 \sim N(M_1, \Sigma_1)$ and $X_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors such that

$$M_1 = \begin{bmatrix} p \\ q \end{bmatrix}, M_2 = \begin{bmatrix} r \\ s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} b^{-1} & 0 \\ 0 & a^{-1} \end{bmatrix}, \tag{7.20}$$

The optimal Bayes classifier is a pair of straight lines *if and only if* $(p-r)^2 = (q-s)^2$, for a, b any positive real numbers. Moreover, if $M_1 = \begin{bmatrix} r \\ s \end{bmatrix}$ and $M_2 = \begin{bmatrix} -r \\ -s \end{bmatrix}$, the condition simplifies to $r^2 = s^2$.

Proof. Multiplying both sides of (6.7) by -1, the discriminant function can be expressed as follows:

$$\begin{bmatrix} x-p \\ y-q \end{bmatrix}^T \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x-p \\ y-q \end{bmatrix} - \begin{bmatrix} x-r \\ y-s \end{bmatrix}^T \begin{bmatrix} b & 0 \\ 0 & a \end{bmatrix} \begin{bmatrix} x-r \\ y-s \end{bmatrix} = 0. \tag{7.21}$$

Note that $\log \frac{|\Sigma_2|}{|\Sigma_1|} = 0$, since $|\Sigma_1| = |\Sigma_2|$.
After performing matrix operations, (7.21) reduces to:

$$a(x-p)^2 + b(y-q)^2 - b(x-r)^2 - a(y-s)^2 = 0. \tag{7.22}$$

By expanding the quadratic terms and applying the distributive law we obtain:

$$ax^2 - 2apx + ap^2 + by^2 - 2bqy + bq^2 - bx^2 + 2brx - br^2 - ay^2 + 2asy - as^2 = 0. \tag{7.23}$$

Taking common factors x and y from the linear and quadratic terms, we obtain:

$$(a-b)x^2 + (b-a)y^2 + 2(-ap+br)x + 2(-bq+as)y + (ap^2+bq^2-br^2-as^2) = 0\,. \tag{7.24}$$

Equation (7.24) is a pair of straight lines *if and only if* the following condition is satisfied [17]:

$$(a-b)(b-a)(ap^2+bq^2-br^2-as^2) - (a-b)(-bq+as)^2 + (a-b)(-ap+br)^2 = 0\,. \tag{7.25}$$

Dividing (7.25) by $(a-b)$ yields:

$$(b-a)(ap^2+bq^2-br^2-as^2) - (-bq+as)^2 + (-ap+br)^2 = 0\,. \tag{7.26}$$

By expanding the quadratic terms and applying the distributive law again, we obtain:

$$\begin{aligned} abp^2 + b^2q^2 - b^2r^2 - abs^2 - a^2p^2 - abq^2 + abr^2 + a^2s^2 \\ -b^2q^2 + 2abqs - a^2s^2 + a^2p^2 - 2abpr + b^2r^2 = 0\,. \end{aligned} \tag{7.27}$$

After canceling some terms, and dividing by ab, (7.27) can be simplified to:

$$(p^2 - 2pr + r^2) - (q^2 - 2qs + s^2) = 0\,. \tag{7.28}$$

We can rewrite (7.28) in quadratic terms as follows:

$$(p-r)^2 = (q-s)^2\,. \tag{7.29}$$

This proves the first assertion.

By setting the means to $M_1 = \begin{bmatrix} r \\ s \end{bmatrix}$ and $M_2 = \begin{bmatrix} -r \\ -s \end{bmatrix}$, (7.29) can be rewritten as:

$$r^2 = s^2. \tag{7.30}$$

The "only if" part of the proof is achieved by following the algebraic steps, in a reverse manner, from (7.29) to (7.21).

Hence the theorem. □

Theorem 7.6 states the necessary and sufficient conditions for a pairwise linear classifier between two normal random vectors, with means and covariances of the form given in (7.20).

7.1.3 Different Covariances and Equal Means

In this section, we discuss a particular instance of the problem discussed in Section 7.1.2. Let us consider the generalization of Minsky's paradox, that is, when $M_1 = M_2$. We shall now show that it is always possible to find a pair of straight lines when $M_1 = M_2$, $\Sigma_1 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}$, and $\Sigma_2 = \begin{bmatrix} b^{-1} & 0 \\ 0 & a^{-1} \end{bmatrix}$, thus resolving the paradox in the most general case.

Theorem 7.7. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors such that:

$$M_1 = M_2 = \begin{bmatrix} r \\ s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} b^{-1} & 0 \\ 0 & a^{-1} \end{bmatrix}, \tag{7.31}$$

The optimal classifier obtained by Bayes classification is a pair of straight lines for positive real numbers a and b, where r and s are any real numbers.

Proof. The proof of this theorem is straightforward using the result of Theorem 7.6.

We know that for Σ_1 and Σ_2, it is possible to find p, q, r, and s, such that $(p-s)^2 = (q-s)^2$. Substituting r for p and s for q, in (7.29), we have $(r-r)^2 = (s-s)^2$. Therefore, for positive real numbers a and b, where r and s are any real numbers, the optimal classifier is a pair of straight lines. □

Clearly, Theorem 7.7 states the necessary and sufficient condition for a pairwise linear classifier between two normal random vectors with covariance matrices of the form given in (7.20), where the means are equal. The power of this will be obvious when the classification results are discussed in a subsequent section.

7.2 Pairwise Classification

7.2.1 The Discriminant Function

In this section, we discuss the pairwise classification with the linear discriminant functions determined in Section 7.1.3, for dimension $d = 2$.

Equations (7.7) and (7.24) are the discriminant functions that represent a pair of straight lines for the cases discussed in Sections 7.1.1 and 7.1.2, respectively. For the purpose of classification, we need to find one equation for each straight line. This is done by inspection or by solving the quadratic equation in terms of y [17]. These second degree polynomial equations have the following roots:

$$y_+ = A_1 x + B_1, \text{ and } y_- = A_2 x + B_2 . \tag{7.32}$$

We now analyze the discriminant functions for the distributions discussed in Sections 7.1.1, 7.1.2, and 7.1.3.

(a) Consider the distributions discussed in Section 7.1.1, i.e. as given in (7.3). By solving for the roots of (7.7), a quadratic polynomial of y, we have[3]:

[3]This was verified by using the symbolic computation package Maple V [22, 34].

$$y_+ = \alpha(a-1)x + \alpha(a+1)r - \beta s, \text{ and} \tag{7.33}$$

$$y_- = \alpha(1-a)x - \alpha(a+1)r - \beta s, \tag{7.34}$$

where $\alpha = \frac{\sqrt{\frac{1-b}{a-1}}}{b-1}$ and $\beta = \frac{b+1}{b-1}$. These now represent the lines that constitute the discriminant.

(b) Let us consider now the third case discussed in Section 7.1.3. The equation for the straight lines are the roots of (7.24). The equation for each straight line can be found as per the following theorem.

Theorem 7.8. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normally distributed random vectors with parameters of the form of (7.31). The equations of the linear discriminant functions for the optimal Bayesian classifiers, are:

$$y_+ = -x + (r+s) \text{ and} \tag{7.35}$$

$$y_- = x + (s-r)\,. \tag{7.36}$$

Proof. Consider Equation (7.24), which specifies the discriminant function between ω_1 and ω_2 with the distributions as stated in Theorem 7.6.

Since $M_1 = M_2 = \begin{bmatrix} r \\ s \end{bmatrix}$, (7.24) can be expressed as follows:

$$(b-a)y^2 + 2(a-b)sy + [(a-b)x^2 + 2(b-a)rx + (a-b)r^2 + (b-a)s^2] = 0\,. \tag{7.37}$$

Being a polynomial of second degree in y, the roots of this equations are given by:

$$\frac{-2(a-b)s \pm \sqrt{4(a-b)^2s^2 - 4(b-a)[(a-b)x^2 + 2(b-a)rx + (a-b)r^2 + (b-a)s^2]}}{2(b-a)}\,. \tag{7.38}$$

After applying the distributive law, and canceling some terms and lengthy manipulations, we can rewrite (7.38) as follows:

$$\frac{-2(a-b)s \pm \sqrt{4(a-b)^2(x-r)^2}}{2(b-a)}. \tag{7.39}$$

After solving the square root expression and canceling the terms $(a-b)$ and $(b-a)$, we have a resulting equation that has two solutions:

$$y_+ = -x + (r+s) \text{ and} \tag{7.40}$$
$$y_- = x + (s-r). \tag{7.41}$$

□

(c) Suppose also that:

$$M_1 = \begin{bmatrix} r \\ s \end{bmatrix} \text{ and } M_2 = \begin{bmatrix} -r \\ -s \end{bmatrix}.$$

We earlier showed that we obtain a pairwise linear discriminant function when $r^2 = s^2$. In this case, the equations of the linear discriminant functions are[4]:

$$\begin{array}{lll} y_+ = -x, & y_- = x - \frac{2(a+b)r}{a-b} & \text{if } s = r, \text{ and} \\ y_+ = x, & y_- = -x + \frac{2(a+b)r}{a-b} & \text{if } s = -r. \end{array} \tag{7.42}$$

7.2.2 Pairwise Linear Classification

The general scheme of a linear classifier is the following [35]. Given a vector, X, the discriminant function as a linear combination of X is:

[4]This was verified by using the symbolic computation package Maple V [22, 34].

$$g(X) = W^T X + w, \tag{7.43}$$

where W is the *weight vector* that gives the direction of the hyperplane, and w is the *threshold weight* that represents the distance from the origin to the hyperplane. We decide ω_1, if $g(x) > 0$, and ω_2, if $g(x) < 0$.

The main problem in this scheme is to find W and w, given the training feature vectors. One of the most popular approaches uses the *perceptron* algorithm that is the basis of the learning algorithm in *neural networks*. One of the disadvantages of this approach is that it uses sub-optimal optimization techniques, such as the *gradient descent* procedure. This procedure can become "stuck" in a local optimum.

In our case, we have the optimal linear classifier, represented by a pair of straight lines, as given by the following equations:

$$g_1(X) = A_1 y + B_1 x + C_1\,, \text{ and} \tag{7.44}$$

$$g_2(X) = A_2 y + B_2 x + C_2\,. \tag{7.45}$$

The weight vectors and the threshold weights are:

$$W_1 = \begin{bmatrix} B_1 \\ A_1 \end{bmatrix}, W_2 = \begin{bmatrix} B_2 \\ A_2 \end{bmatrix}, w_1 = C_1, \text{ and } w_2 = C_2\,. \tag{7.46}$$

We can write the general inequality for classification with a pair of hyperplanes as follows:

$$g_1(X) g_2(X) \underset{\omega_2}{\overset{\omega_1}{\lessgtr}} 0\,, \tag{7.47}$$

where $g_1(X) = W_1^T X + w_1$ and $g_2(X) = W_2^T X + w_2$.

The classification is done by using the following scheme:

- $g_1(X) < 0$ and $g_2(X) < 0 \Rightarrow X \in \omega_2$,
- $g_1(X) < 0$ and $g_2(X) > 0 \Rightarrow X \in \omega_1$,
- $g_1(X) > 0$ and $g_2(X) < 0 \Rightarrow X \in \omega_1$, and
- $g_1(X) > 0$ and $g_2(X) > 0 \Rightarrow X \in \omega_2$.

Consider Equation (7.36). This can be written as follows:

$$y - x + (r - s) = 0 . \tag{7.48}$$

Dividing Equation (7.37) by Equation (7.48) yields:

$$(b - a)[y + x - (r + s)] = 0 . \tag{7.49}$$

Note that we do not cancel the term $b - a$, because we are looking for a classifier that can be obtained by replacing the equality symbol in the discriminant function by the symbol $\lessgtr$, and $b - a$ can be positive or negative. Next, we divide (7.49) by $|b - a|$ and multiply it by $sgn(b-a)$, where $sgn(b-a)$ is 1 if $b-a$ is positive and -1 otherwise.

The discriminant functions in vectorial form are:

$$g_1(X) = sgn(b - a) \left\{ \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + (-r - s) \right\}, \text{ and} \tag{7.50}$$

$$g_2(X) = \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + (r - s) . \tag{7.51}$$

In an analogous manner, the discriminant functions, $g_1(X)$ and $g_2(X)$, and the classification scheme for equations given in (7.40), (7.41), and (7.42) can be obtained.

7.3 Simulation Results on Synthetic Data

In this section, we present some examples illustrating the different cases discussed in previous sections. In all the examples, we have chosen the dimension $d = 2$ and two classes, ω_1 and ω_2. We also discuss the empirical results obtained after testing the linear classifier with 100 points for each class generated randomly using the *maximum likelihood* approach in estimating the parameters [146], assuming that they are of the form found in the respective cases.

The two classes, ω_1 and ω_2, are represented by two normal random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$, respectively. We used two instances of each of the three cases and generated a set of 100 normal random points, in order to test the accuracy of the classifiers.

Example 7.1 (Pairwise Linear Discriminants in Diagonalized Classes). In the first test (referred to as DD-1 and DD-2) we considered the pairwise linear discriminant function in diagonalization. We used the following covariance matrices and mean vectors (estimated from 100 training samples) to yield the respective classifier.

DD-1: $M_1 = -M_2 \approx \begin{bmatrix} 1.0342 \\ 1.8686 \end{bmatrix}, \Sigma_1 = I, \Sigma_2 \approx \begin{bmatrix} .4599 & 0 \\ 0 & 2.8232 \end{bmatrix}$

DD-2: $M_1 = -M_2 \approx \begin{bmatrix} -1.4602 \\ -1.1913 \end{bmatrix}, \Sigma_1 - = I, \Sigma_2 \approx \begin{bmatrix} 1.8395 & 0 \\ 0 & .4278 \end{bmatrix}$

The plots of the points and the linear discriminant functions are depicted in Figures 7.1 and 7.2 respectively. The accuracy of the classifier was 99% for ω_1 and ω_2 in the first test set, DD-1, and 94% for ω_1 and 93% for ω_2 in the second test set, DD-2. The power of the scheme is obvious! □

Example 7.2 (Pairwise Linear Discriminants with Different Means). In the second test (referred to as DM-1 and DM-2) we considered the pairwise linear discriminant with different means. The following estimated covariance matrices and mean vectors were obtained by using 100 training samples.

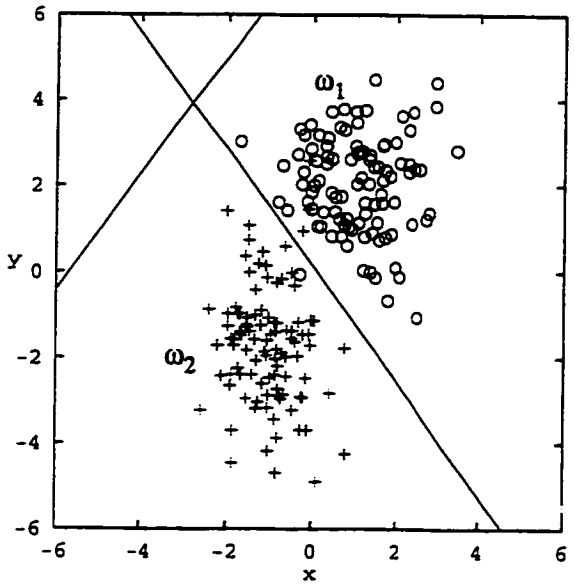


Figure 7.1: Example of pairwise linear discriminant in diagonalization for the case described in Theorem 7.2. The data set is DD-1.

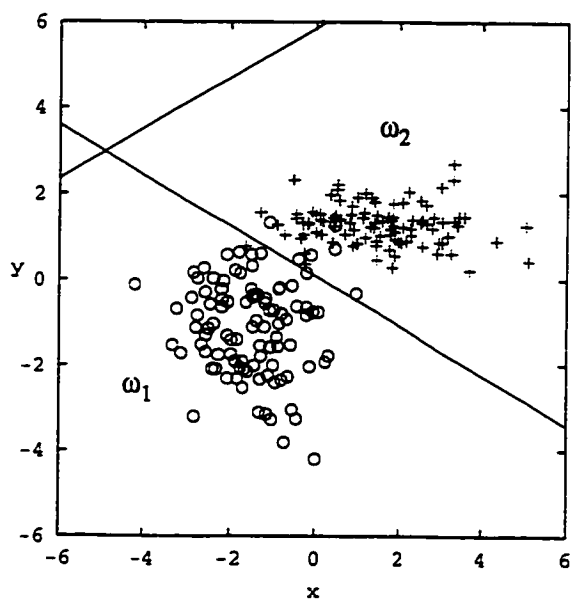


Figure 7.2: Example of pairwise linear discriminant in diagonalization for the case described in Theorem 7.2. The data set is DD-2.

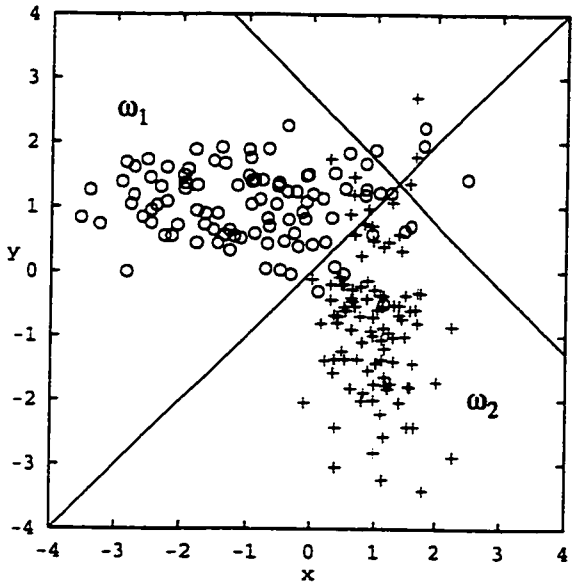


Figure 7.3: Example of pairwise linear discriminant with different means for the case described in Theorem 7.6. The data set is DM-1.

$$\textbf{DM-1: } M_1 = -M_2 \approx \begin{bmatrix} -.9555 \\ .9555 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} 1.8077 & 0 \\ 0 & .3188 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} .3188 & 0 \\ 0 & 1.8077 \end{bmatrix}$$

$$\textbf{DM-2: } M_1 = -M_2 \approx \begin{bmatrix} .8293 \\ .8293 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} .3615 & 0 \\ 0 & 2.5503 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} 2.5503 & 0 \\ 0 & .3615 \end{bmatrix}$$

Using the above parameters, the pairwise linear classifier was derived. The plots of the points and the linear discriminant functions are shown in Figures 7.3 and 7.4 respectively. The accuracy of the classifier was 91% for ω_1 and 95% for ω_2 in the first test set, DM-1, and 88% for ω_1 and ω_2 in the second test set, DM-2. □

Example 7.3 (Pairwise Linear Discriminants with Equal Means). To show the power of the scheme, we also tested our results for the case of the pairwise linear classifier with equal means (EM-1 and EM-2) for the generalized Minsky's Paradox. By using 100 training samples generated with equal means but mirrored covariances, we obtained the following estimated covariance matrices and mean vectors.

$$\textbf{EM-1: } M_1 = M_2 \approx \begin{bmatrix} -1.9394 \\ 4.1875 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} 4.5193 & 0 \\ 0 & .255 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} .255 & 0 \\ 0 & 4.5193 \end{bmatrix}$$

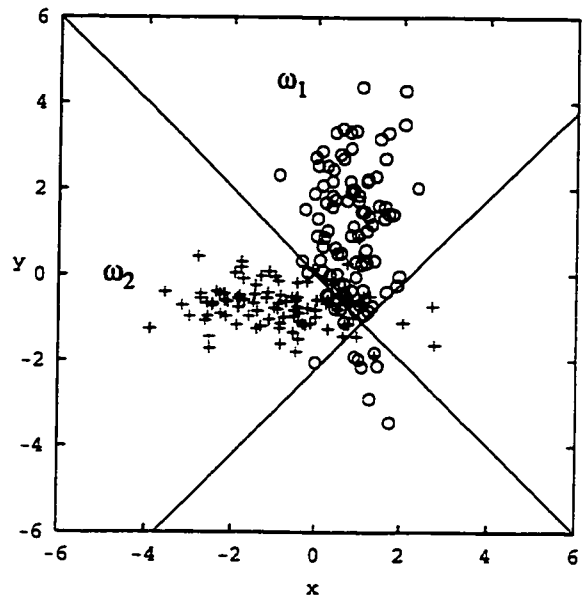


Figure 7.4: Example of pairwise linear discriminant with different means for the case described in Theorem 7.6. The data set is DM-2.

$$\textbf{EM-2:}\ M_1 = M_2 \approx \begin{bmatrix} 5.9841 \\ 6.1766 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} .0904 & 0 \\ 0 & 12.7515 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} 12.7515 & 0 \\ 0 & .0904 \end{bmatrix}$$

The plots of the points and the linear discriminant functions from these estimates are given in Figures 7.5 and 7.6 respectively. The accuracy of the classifier was 82% for ω_1 and 85% for ω_2 in the first test set, EM-1, and 96% for ω_1 and 90% for ω_2 in the second test set, EM-2. □

The empirical results of classification are given in Table 7.1. The first column corresponds to the test case. The second and third columns represent the percentage of correctly classified points belonging to ω_1 and ω_2, respectively. Note the very high accuracy in the first two cases, DD-1 and DD-2, which correspond to the pairwise linear discriminant in diagonalization case. DM-1 and DM-2 are "worse" in accuracy, but it is still high. These correspond to the pairwise linear discriminant with different means. The sixth and seventh rows correspond to the case where the means are identical, referred to as EM-1 and EM-2. The accuracy is very high in this case, despite the fact that the classes overlap and the discriminant functions are pairwise linear. The power

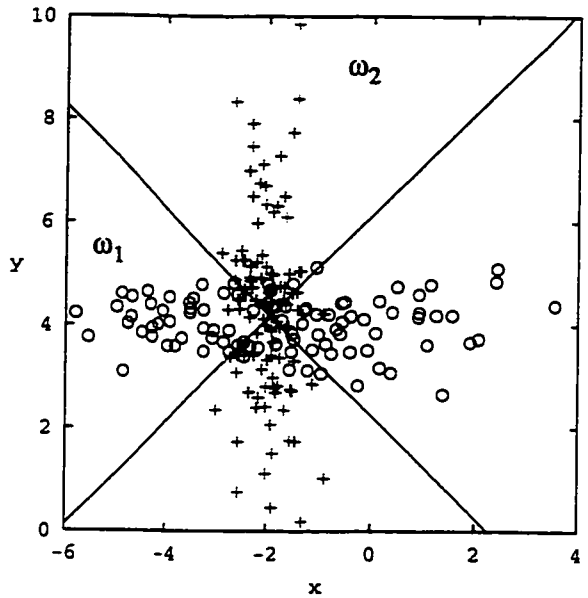


Figure 7.5: Example of pairwise linear discriminant with equal means for the case described in Theorem 7.7. The data set is EM-1.

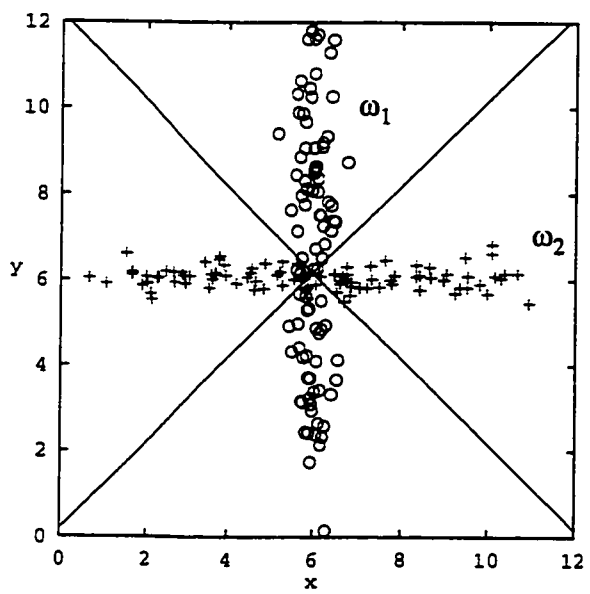


Figure 7.6: Example of pairwise linear discriminant with equal means for the case described in Theorem 7.7. The data set is EM-2.

Example	accuracy for ω_1	accuracy for ω_2
DD-1	98 %	99 %
DD-2	95 %	94 %
DM-1	91 %	95 %
DM-2	88 %	88 %
EM-1	82 %	85 %
EM-2	96 %	90 %

Table 7.1: Accuracy in classification of test points generated with the six examples presented above. The accuracy is given in percentage of points correctly classified.

of the results presented are again obvious and the resolution of Minsky's Paradox is clear.

7.4 Pairwise Linear Classifiers in Real Life Data

As shown earlier, there are three cases in which the optimal classifier is pairwise and linear. The most general case is the one presented in Section 7.1.1, for which the condition is given in (7.4), and the parameters are of the form of (7.3). Real life data sets that satisfy these constraints are not very common, and in general, a classification scheme should be applicable to any data set (or eventually to a particular domain). In the next subsection, we propose a method that substitutes the actual parameters of the data sets by approximated parameters for which the required constraints are satisfied. This, in turn, reduces to solving a constrained optimization problem. We also present the empirical results of the experiments that we have conducted on real life data.

7.4.1 Parameter Approximation

To solve the constrained optimization problem alluded to above, we propose a method that finds approximate parameters for normal distributions that satisfy the constraints of (7.3) and (7.4). The problem and its solution are presented below.

Suppose that we are given two data sets , $\mathcal{D}_1 = \{X_{1_1}, X_{1_2}, \ldots, X_{1_{N_1}}\}$ and $\mathcal{D}_2 = \{X_{2_1}, X_{2_2}, \ldots, X_{2_{N_2}}\}$, which contain labeled samples, where X_{1_j} and X_{2_j} are drawn independently from their respective classes. We assume that the two classes corre-

sponding to these samples are represented by two normally distributed random vectors $\mathbf{X}_1$ and $\mathbf{X}_2$, whose parameters, $\theta_1 = \begin{bmatrix} M_1 \\ \Sigma_1 \end{bmatrix}$ and $\theta_2 = \begin{bmatrix} M_2 \\ \Sigma_2 \end{bmatrix}$, are to be estimated. Our aim is to find the *maximum-likelihood* estimates $\hat{\theta}_1$ and $\hat{\theta}_2$ that satisfy the constraints stated in (7.3) and (7.4).

We, first of all, use the maximum-likelihood method to estimate the parameters from $\mathcal{D}_1$ and $\mathcal{D}_2$, i.e. M_1, M_2, Σ_1 and Σ_2 [38]. By shifting the origin such that $M_1 = -M_2$, and performing the simultaneous diagonalization process discussed earlier, we obtain two transformed data sets, $\mathcal{D}_1'$ and $\mathcal{D}_2'$, as follows:

$$X_{i_j}' = \Psi_2^T \Lambda_1^{-\frac{1}{2}} \Phi_1^T \left\{ X_{i_j} - \left[M_2 + \frac{1}{2}(M_1 - M_2) \right] \right\}, \; i = 1, 2, \tag{7.52}$$

where Φ_1 is the eigenvector matrix of Σ_1, Λ_1 is the eigenvalue matrix of Σ_1, and Ψ_2 is the eigenvector matrix of Σ_{2_Z} obtained after performing the transformation $\mathbf{Z}_i = \Lambda_1^{-\frac{1}{2}} \Phi_1^T \mathbf{X}_i$, for $i = 1, 2$.

To clarify issues regarding the notation used here, we use "primed" variables with the "′" symbol to denote parameters or samples after the transformation of (7.52).

Once the samples are transformed into the new space, we proceed with the *constrained* maximum-likelihood estimation problem. The aim is to find the parameters $\{\hat{\theta}_i'\}$ that maximize the likelihood of $\{\theta_i\}$ with respect to the samples $\{\mathcal{D}_i'\}$, for $i = 1, 2$,

$$P(\mathcal{D}_i' | \theta_i') = \prod_{j=1}^{N_i} P(X_{i_j}' | \theta_i') \,, \tag{7.53}$$

while satisfying (7.3) and (7.4). This is equivalent to maximizing the *log-likelihood* function

$$l(\theta_i') = \log P(\mathcal{D}_i' | \theta_i') \,. \tag{7.54}$$

From (7.52), it can be seen that if $M_1 = -M_2$, then for $i = 1, 2$,

$$M_i' = \Psi_2^T \Lambda_1^{-\frac{1}{2}} \Phi_1^T \left\{ M_i - \left[M_2 + \frac{1}{2}(M_1 - M_2) \right] \right\} \tag{7.55}$$

have the form $M_1' = -M_2'$, and hence our problem is to find Σ_1' and Σ_2' that maximize the likelihood function. Since, after the transformation, $\Sigma_1' = I$, we are left with the task of determining Σ_2' which we assume is diagonal, and which has the form $\Sigma_2' = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}$. Thus, (7.54) can be written as follows:

$$l(\theta_2') = \sum_{j=1}^{N_2} \log P(X_{2_j}' | \theta_2') \,. \tag{7.56}$$

Substituting $P(X_{2_j}' | \theta_2')$ for the normal density function, and assuming that M_1' and M_2' have the form $M_1' = -M_2' = [r, s]^T$, we have the following constrained optimization problem:

Determine the value of $\hat{\Sigma}_2'$ that maximizes

$$l(\theta_2') = \sum_{j=1}^{N_2} \left\{ -\frac{1}{2} \log \left[(2\pi)^2 |\Sigma_2'| \right] - \frac{1}{2} (X_{2_j}' - M_2')^T (\Sigma_2')^{-1} (X_{2_j}' - M_2') \right\} , \tag{7.57}$$

subject to the constraint

$$g(\theta_2') = a(1-b)r^2 + b(1-a)s^2 - \frac{1}{4}(ab - a - b + 1) \log(ab) = 0 \,. \tag{7.58}$$

Using the Lagrange multiplier, this *constrained* optimization problem can be transformed into an *unconstrained* optimization problem, for which we have to find the solutions to the following system of equations. After the corresponding differentiation and some algebraic manipulations, and assuming that X_{2_j}' has the form $X_{2_j}' = \begin{bmatrix} x_{2_j}' \\ y_{2_j}' \end{bmatrix}$,

we get:

$$\begin{aligned}
&\tfrac{1}{2}\left[\tfrac{N_2}{a} - \textstyle\sum_{j=1}^{N_2}(x'_{2_j} + r)^2\right] + \\
&\lambda\left[(1-b)r^2 - bs^2 - \tfrac{1}{4}(b-1)\log(ab) - \tfrac{1}{4a}(ab-a-b+1)\right] &= 0 \quad (7.59)\\
&\tfrac{1}{2}\left[\tfrac{N_2}{b} - \textstyle\sum_{j=1}^{N_2}(y'_{2_j} + s)^2\right] + \\
&\lambda\left[-ar^2 + (1-a)s^2 - \tfrac{1}{4}(a-1)\log(ab) - \tfrac{1}{4b}(ab-a-b+1)\right] &= 0 \quad (7.60)\\
&a(1-b)r^2 + b(1-a)s^2 - \frac{1}{4}(ab-a-b+1)\log(ab) &= 0 \quad (7.61)
\end{aligned}$$

The system of equations given above has no closed-form algebraic solution for a, b and λ. Instead, it can be solved numerically for $a > 0$, $b > 0$, and where λ is a real number. The numerical solution that satisfies the constraints given in Theorem 7.3 are indeed $\hat{a}$ and $\hat{b}$. Using r, s, $\hat{a}$ and $\hat{b}$, the linear classifier is thus obtained from (7.33) and (7.34).

7.4.2 Empirical Results

In this section, we discuss the empirical results that we obtained after performing classification tasks using the approximated optimal (pairwise) linear classifier on a real life data set drawn from the UCI machine learning repository[5], namely the Wisconsin Diagnostic Breast Cancer (WDBC) data set. It contains two data sets, one for the "benign" class and the other for the "malignant" class. Each sample contains 30 real-valued features representing the radius, texture, perimeter, area, smoothness, compactness, etc., computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. Details of these features, and how they are obtained can be found in [90].

The benign class and the malignant class data sets contain 357 and 212 samples respectively. From these data sets, we have randomly selected 100 samples for training, and 100 samples for the testing of each class. The training samples are different from those used in the testing. Thus, we have used the *two-folded cross validation* approach [38].

[5]Available electronically at http://www.ics.uci.edu/~mlearn/MLRepository.html.

Classifier→ Class ↓	Fisher's	Pairwise
Benign	96%	98%
Malignant	91%	93%

Table 7.2: Accuracy in classification obtained from the Fisher's discriminant approach, and the approximated optimal pairwise linear classifier on the WDBC data set.

For the training and classification tasks we have composed 15 data subsets with all possible pairs of features obtained from the first six features. Our classifier was trained by following the procedure described in Subsection 7.4.1, thus yielding the approximated pairwise linear classifier for each subset. The classification task was performed using these linear classifiers, and a voting scheme was invoked. This scheme assigned a class to a testing sample for each of the 15 data subsets, and subsequently classified the sample to the class that yielded positive classification for eight or more voters.

To compare our result with a well-known scheme, we have also trained and performed classification tasks using the Fisher's discriminant approach [38], and the same voting scheme on the same 15 data subsets.

The empirical results corresponding to the accuracy obtained in classification are shown in Table 7.2. Observe the superiority of the approximated pairwise linear classifier over Fisher's discriminant. This again demonstrates the power of our scheme. This is further clarified from another perspective in Figure 7.7, where we plot the "area" and "smoothness" features.

The symbol '$\circ$' and '$+$' correspond to the benign class and the malignant class testing points respectively. These points were obtained after performing the transformation of (7.52). The circle and the ellipses correspond to the points with the same Mahalanobis distance (unity in this case), for the benign and the malignant classes respectively. Σ_2 is the covariance matrix obtained by using the *standard* maximum-likelihood estimate, and Σ_2' is the one obtained using the *constrained* maximum-likelihood introduced here. The parabola represents the optimal quadratic classifier obtained from the parameters estimated using the standard maximum-likelihood estimate. Observe that

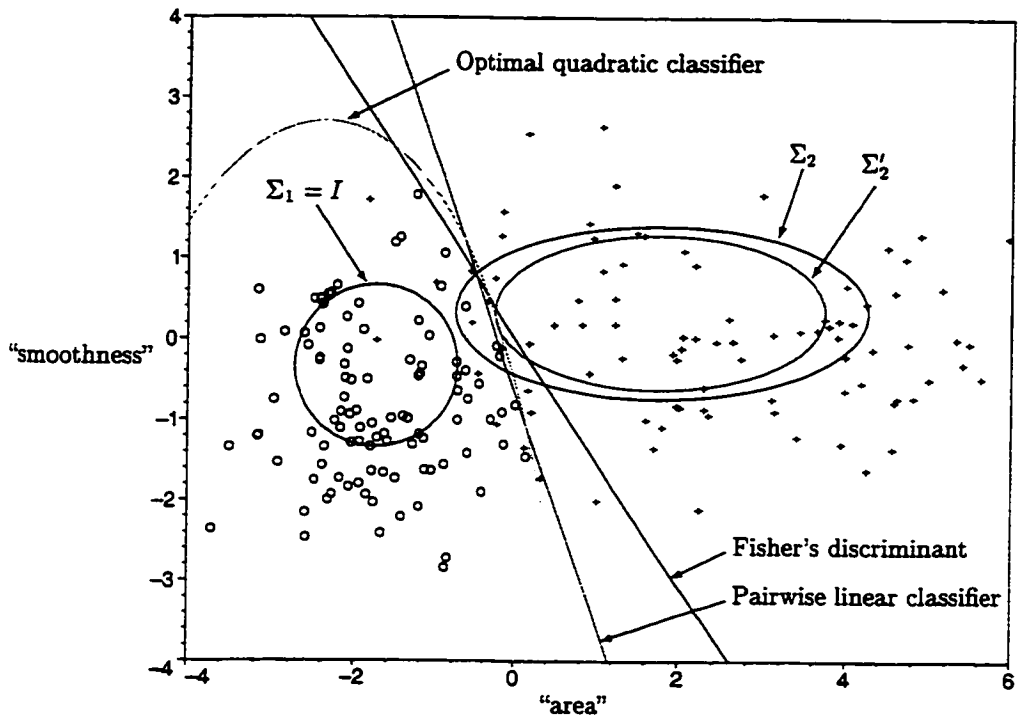


Figure 7.7: Optimal quadratic classifier, Fisher's discriminant function, and the approximated pairwise linear classifier used to classify the "area" and "smoothness" features from the WDBC data set.

some points of the malignant class are misclassified when using Fisher's discriminant. Although this is only one of the 15 classification tasks that we have performed, the superiority of our scheme over traditional linear classifiers (such as the Fisher's discriminant approach) is again corroborated. Observe that our linear classifier is closer to the optimal than Fisher's discriminant in the area in which the samples are more likely to occur. Observe also that although our classifier is of the form of a pair of straight lines (one of them is not shown because it is outside the ranges of the graph), only the one shown in the figure can be used. In spite of not utilizing the full capabilities of the pairwise classifier, we still maintain the superior classification accuracy.

We conclude this section by observing that although we have performed the classification tasks in the transformed space, this can be done in the original space by avoiding the burden of transforming each individual sample. In fact, by performing the inverse transformation on the discriminant functions, the classification can be done

in the original space, and the classifiers are still linear. The details of this are omitted as they are straightforward.

7.5 Conclusions

In this chapter, we have shown that it is possible to find the optimal linear discriminant function for normally distributed features even though the covariance matrices are different. In all the cases discussed here, the function obtained is a *pair* of straight lines, which is a particular case of the solution to a second degree general equation.

By a formal procedure, we have determined the conditions for these particular discriminant functions in three cases. The first case occurs after diagonalization. Having two classes with normal distribution and any covariance matrices, we can always perform diagonalization, and obtain the required covariance matrices. We have explicitly derived the necessary and sufficient conditions for the covariance matrices and the mean vectors so as to yield a pair of straight lines for the optimal classifier.

The second case is when we have particular forms in the two diagonal covariance matrices. In this case, the two terms of the diagonal of one matrix have to be a permutation of the two terms of the other matrix. For this case we have again derived the necessary and sufficient conditions for an optimal pairwise linear discriminant function.

In the third case, assuming equal means, we have found that it is always possible to obtain a pair of straight lines when we have covariance matrices with the same form as found in the second case. This resolves Minsky's paradox!

The empirical results obtained show that the accuracy of the classifier is very high. This is understandable since the classifier is optimal. The degree of this accuracy is even more amazing when we recognize that we are dealing with a linear discriminant function for classes which are significantly overlapping.

We have also presented a scheme from which the maximum likelihood pairwise linear classifier can be estimated. The superiority of this approach over the traditional Fisher's discriminant approach has been experimentally verified on a real life data set, namely the Wisconsin Diagnostic Breast Cancer data set. This superiority has been demonstrated numerically and graphically. The generalized results for the d-dimensional case are presented in Chapter 8.

Chapter 8

d-Dimensional Optimal Pairwise Linear Classifiers

In Chapter 7, we showed that although the general classifier for two dimensional normally distributed random vectors is a second-degree polynomial, in various cases, this polynomial degenerates to be either a single straight line or a pair of straight lines. Thus, as opposed to the traditional results, we showed that the classifier can be linear even when the covariance matrices are not equal. As we showed, the discriminant function is a pair of first-order equations, which are factors of the second-order polynomial (i.e. the discriminant function). When the factors are equal, the discriminant function is given by a single straight line, which corresponds to the traditional case when the covariance matrices are equal.

In this chapter, we extend these results and present the formal theory for optimal pairwise linear classifiers for d-dimensional normally distributed random vectors, where $d > 2$. We assume that the features of an object to be recognized are represented as a d-dimensional random vector characterized by a probability distribution function. We deal only with the case in which these random vectors have a jointly normal distribution, where class ω_i has a mean M_i and covariance matrix Σ_i.

Without loss of generality, we again assume that the classes ω_1 and ω_2 have the same *a priori* probability, 0.5, in which case, the discriminant function is given by (6.7). When $\Sigma_1 = \Sigma_2$ the discriminant function is linear [38, 47, 78, 125, 146, 162]. For the

case when Σ_1 and Σ_2 are arbitrary, the optimal classifier is a general equation of second degree which results in the discriminant being a hyperparaboloid, a hyperellipsoid, a hypersphere, a hyperboloid, or a pair of hyperplanes. The latter case is the focus of the study that we present in this chapter.

As in Chapter 7, we present here the formal theory for three different cases for the form of the parameter distribution of the classes. The first one is when the two random vectors representing the classes are diagonalized, i.e. the covariance matrices are the identity and a diagonal matrix. In the second case, the mean vectors are different and the covariance matrices are diagonal. The third case is when the mean vectors are equal, and the covariance matrices have to satisfy a special form. The latter resolves the generalized multi-dimensional Minsky's paradox.

The formal analysis of optimal pairwise linear classifiers for multi-dimensional random vectors discussed in this chapter has been published in [134, 135, 139].

8.1 Multi-Dimensional Pairwise Classifiers for Diagonalized Classes

In Section 7.1.1, we present the necessary and sufficient conditions required so that the optimal classifier is a pair of straight lines, for two dimensional normal random vectors obtained by performing diagonalization. Using these results, we present here the cases for the d-dimensional scenario in which the optimal Bayes classifier is a pair of hyperplanes.

Let us consider now the more general case for $d > 2$. Using the results mentioned above, we derive the necessary and sufficient conditions for an optimal pairwise linear classifier. From the inequality constraints stated in Cases I and II of Theorem 7.3, we state and prove that it is not possible to find the optimal Bayes classifier as a pair of hyperplanes for these conditions when $d > 2$. We modify the notation marginally. We use the symbols $(a_1^{-1}, a_2^{-1}, \ldots, a_d^{-1})$ to synonymously refer to the marginal variances $(\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2)$.

Theorem 8.1. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normally distributed random vectors, such that

$$M_1 = -M_2 = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_d \end{bmatrix}, \Sigma_1 = I, \text{ and } \Sigma_2 = \begin{bmatrix} a_1^{-1} & 0 & \dots & 0 \\ 0 & a_2^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_d^{-1} \end{bmatrix} \tag{8.1}$$

where $a_i \neq 1$, $i = 1, \dots, d$. There are no real numbers m_i, $i = 1, \dots, d$, such that the optimal Bayes classifier is a pair of hyperplanes.

Proof. In order to obtain a pair of hyperplanes for the optimal classifier in the d-dimensional space, the projection of these hyperplanes onto the plane determined by the axes x_i and x_j must be a pair of straight lines for all $i, j = 1, \dots, d$, $i \neq j$.

Without loss of generality[1], suppose that the conditions stated in Theorem 7.3 are satisfied for the axes x_i and x_j. Now, consider another axis, x_k, $k \neq i$, $k \neq j$, and let us analyze the possibility of finding a pair of straight lines, which is the optimal classifier between the axes x_k and x_i, and between the axes x_k and x_j.

- Suppose that $0 < a_i < 1$ and $a_j > 1$.

  In this case, if $0 < a_k < 1$, using the result of Theorem 7.3, we see that it is impossible to find the corresponding real numbers r_{ik} and s_{ik} such that the optimal classifier is a pair of straight lines between the axes x_k and x_i, since $0 < a_i < 1$.

  On the other hand, if $a_k > 1$, again, there are no real numbers r_{jk} and s_{jk} such that the optimal classifier is a pair of straight lines between the axes x_k and x_j, since $a_j > 1$.

- Suppose that $a_i > 1$ and $0 < a_j < 1$.

[1] We refer to the quantities r and s of Theorem 7.3 as r_{ij} and s_{ij}, if the random variables considered are x_i and x_j.

Again, in this case, if $0 < a_k < 1$, it is not possible to find real numbers r_{jk} and s_{jk} so as to yield a pair of straight lines between the axes x_k and x_j, since $0 < a_j < 1$.

Analogously, if $a_k > 1$, it is impossible to find a pair of straight lines between the axes x_k and x_i, since $a_i > 1$.

We have thus shown that it is not possible to find a pair of hyperplanes when $a_i \neq 1, a_j \neq 1$, and $a_k \neq 1$. The result follows. □

Informally speaking, the proof of Theorem 8.1 is accomplished by checking if there is an optimal pairwise linear classifier for all the pairs of axes. This is not possible since, if the condition has to be satisfied, when the first element on the diagonal is less than unity, the second one must be greater than unity. Consequently, there is no chance for a third element to satisfy this condition, pairwise, with the first two elements.

Using the results of Theorem 7.4, we now analyze the possibility of finding the optimal pairwise linear classifiers for the d-dimensional case when some of the entries in Σ_2 are unity.

Theorem 8.2. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normally distributed random vectors with parameters of the form of (8.1). If there exists i such that $a_i \neq 1$, and $a_j = 1$, $m_j = 0$, for $j = 1, \ldots, d$, $i \neq j$, then the optimal Bayes classifier is a pair of hyperplanes.

Proof. In order to obtain a pair of hyperplanes for the optimal classifier in the d-dimensional space, the projection of these hyperplanes onto the plane determined by the axes x_i and x_j must be a pair of straight lines for all $i, j = 1, \ldots, d$, $i \neq j$.

We now invoke the results of Theorem 7.4. Without loss of generality, suppose that $a_i \neq 1$. We analyze the possibility of finding a pair of straight lines between the axes x_i and x_j, $j = 1, \ldots, d$, $i \neq j$.

According to condition (b) of Theorem 7.4, the optimal classifier is a pair of straight lines if $a_i \neq 1$, $a_j = 1$, and $m_j = 0$. To satisfy this condition for all $j = 1, \ldots, d$, $i \neq j$, we need all $a_j = 1$, $m_j = 0$. This is satisfied using condition (c) of Theorem 7.4.

On the other hand, we must check that the optimal classifier is a pair of straight lines between the axes x_j and x_k, $j, k = 1, \ldots, d$, $j \neq k \neq i$. This is clearly satisfied since $a_j = a_k = 1$, which is the traditional case where the optimal classifier is a single straight line (or two coincident straight lines) in the x_j–x_k plane.

Hence the Theorem! □

We now combine the results of Theorems 7.3 and 7.4, and state more general necessary and sufficient conditions to find a pair of hyperplanes as the optimal Bayes classifier. We achieve this using the inequality and equality constraints of these theorems.

The main difference between Theorem 8.2 and the theorem given below is that in the former, all the elements but one of the diagonal of Σ_2 are equal to unity. In the theorem presented below, there are two elements of the diagonal of Σ_2 which are not equal to unity, and therefore they must satisfy (8.1) and the conditions stated in either Case (*i*) or Case (*ii*) of Theorem 7.3.

Theorem 8.3. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normally distributed random vectors with parameters of the form of (8.1). The optimal Bayes classifier is a pair of hyperplanes if there exist i and j such that any of the following conditions are satisfied:

(a) $0 < a_i < 1$, $a_j > 1$, $a_k = 1$, $m_k = 0$, for all $k = 1, \ldots, d$, $k \neq i$, $k \neq j$, with

$$a_i(1 - a_j)m_i^2 + a_j(1 - a_i)m_j^2 - \frac{1}{4}(a_i a_j - a_i - a_j + 1)\log a_i a_j = 0\,. \tag{8.2}$$

(b) $a_i \neq 1$, $a_j = 1$, $m_j = 0$, for all $j \neq i$.

(c) $a_i = 1$, for all $i = 1, \ldots, d$.

Proof. As in Theorem 8.2, to obtain a pair of hyperplanes for the optimal classifier in the d-dimensional space, the projection of these hyperplanes onto the plane determined by the axes x_i and x_j must be a pair of straight lines for all $i, j = 1, \ldots, d$, $i \neq j$.

Without loss of generality, suppose that there exists an index i such that $0 < a_i < 1$. Consider another axis, x_j, $i \neq j$. We have to analyze the following cases for a_j:

(*i*) $0 < a_j < 1$: According to the conditions stated in Theorem 7.3, we cannot have a pairwise linear optimal classifier.

(*ii*) $a_j > 1$: If (8.2) is satisfied between x_i and x_j, for all the x_k, $k = 1, \ldots, d$, $k \neq i$, $k \neq j$, $i \neq j$, the optimal classifier between the axes x_i and x_k, x_j and x_k, is a pair of straight lines only if $a_k = 1$ and $m_k = 0$, using the equality constraint (b) of Theorem 7.4. Hence condition (a) is satisfied.

(*iii*) $a_j = 1$: which is the case of condition (b). Here, using the result of Theorem 8.2, an optimal pairwise linear classifier results if $a_j = 1$ and $m_j = 0$, for all $j = 1, \ldots, d$, $j \neq i$.

The final case considered in condition (c) corresponds to the traditional case in which the optimal Bayes classifier is a single hyperplane when both the covariance matrices are identical.

Hence the theorem. □

8.2 Linear Discriminants with Different Means

In Chapter 7, we have shown that given two normally distributed random vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, with mean vectors and covariance matrices of the form:

$$M_1 = -M_2 = \begin{bmatrix} r \\ s \end{bmatrix}, \Sigma_1 = \begin{bmatrix} a^{-1} & 0 \\ 0 & b^{-1} \end{bmatrix}, \text{ and } \Sigma_2 = \begin{bmatrix} b^{-1} & 0 \\ 0 & a^{-1} \end{bmatrix}, \tag{8.3}$$

the optimal Bayes classifier is a pair of straight lines when $r^2 = s^2$, where a and b are any positive real numbers. The discriminant function for this case is given by:

$$a(x-r)^2 + b(y-s)^2 - b(x+r)^2 - a(y+s)^2 = 0\,. \tag{8.4}$$

We consider now the more general case for $d > 2$. We are interested in finding

the conditions that guarantee a pairwise linear discriminant function. This is given in Theorem 8.4 below.

Theorem 8.4. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors such that

$$M_1 = [m_1 \ldots m_i \ldots m_j \ldots m_d]^T, \; M_2 = [m_1 \ldots m_{i-1} - m_i m_{i+1} \ldots m_{j-1} - m_j m_{j+1} \ldots m_d]^T \tag{8.5}$$

$$\Sigma_1 = \begin{bmatrix} a_1^{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_i^{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_j^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_d^{-1} \end{bmatrix}, \; \Sigma_2 = \begin{bmatrix} a_1^{-1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_j^{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_i^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_d^{-1} \end{bmatrix} \tag{8.6}$$

The optimal classifier, obtained by Bayes classification, is a pair of hyperplanes when

$$m_i^2 = m_j^2 . \tag{8.7}$$

Proof. We shall prove the theorem by utilizing some of the results of the two dimensional case mentioned above. The discriminant function given in (6.7) can be written as follows:

$$(X - M_1)^T \Sigma_1^{-1} (X - M_1) - (X - M_2)^T \Sigma_2^{-1} (X - M_2) = 0 . \tag{8.8}$$

Note that $\log \frac{|\Sigma_2|}{|\Sigma_1|} = 0$, since $|\Sigma_1| = |\Sigma_2|$. After applying some simplifying operations to (8.8), the term $a_k(x_k - m_k)^2$ can be canceled along with its correspondent negative term $-a_k(x_k - m_k)^2$, for $k = 1, \ldots, d$, $k \neq i \neq j$. Equation (8.8) now reduces to:

$$a_i(x_i - m_i)^2 + a_j(x_j - m_j)^2 - a_j(x_i + m_i)^2 - a_i(x_j + m_j)^2 = 0\,. \tag{8.9}$$

Equation (8.9) is the pairwise multi-dimensional version of (8.4). By choosing $a = a_i$, $b = a_j$, $r = m_i$, $s = m_j$, and after rather lengthy algebraic operations, it can be seen that the discriminant function is a pair of hyperplanes whenever:

$$m_i^2 = m_j^2\,. \tag{8.10}$$

□

Theorem 8.4 can be interpreted geometrically as follows. Whenever we have two covariance matrices that differ only in two elements of their diagonal; and whenever the two elements in the second covariance matrix are a permutation of the same rows in the first matrix, if the mean vectors differ only in these two elements, the resulting discriminant function is a pair of hyperplanes.

Indeed, by performing a projection of the space in the x_i and x_j axes, we observe that the discriminant takes on exactly the same shape as that which is obtained from the distribution given in (8.3). Thus effectively, we obtain a pair of straight lines in the two dimensional space from the projection of the pair of hyperplanes in the d-dimensional space.

8.3 Linear Discriminants with Equal Means

We consider now a particular instance of the problem discussed in Section 8.2, which leads to the resolution of the generalization of the d-dimensional Minsky's paradox. In this case, the covariance matrices have the form of (8.6), but the mean vectors are the same for both classes. We shall show now that, with these parameters, it is always possible to find a pair of hyperplanes, which resolves Minsky's paradox in the most general case.

Theorem 8.5. Let $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$ be two normal random vectors, where $M_1 = M_2 = [m_1, \ldots, m_d]^T$, and Σ_1 and Σ_2 have the form of (8.6). The optimal classifier, obtained by Bayes classification, is a pair of hyperplanes.

Proof. From Theorem 8.4, we know that when the mean vectors and covariance matrices have the form of (8.5) and (8.6), respectively, then the optimal Bayes classifier is a pair of hyperplanes if $m_i^2 = m_j^2$. Since $M_1 = M_2$, then $m_i^2 = m_j^2$, for $j = 1, \ldots, d$, $i \neq j$. Hence the optimal Bayes classifier is a pair of hyperplanes. The theorem is thus proved. □

8.4 Simulation Results

In order to test the accuracy of the pairwise linear discriminants, and to verify the results derived here, we have performed some simulations for the different cases discussed above. We have chosen the dimension $d = 3$, since it is easy to visualize and plot the corresponding hyperplanes. In all the simulations, we trained our classifier using 100 randomly generated training samples (which were three dimensional vectors from the corresponding classes). Using the *maximum likelihood* estimation method [146], we then approximated the mean vectors and covariance matrices for each of the three cases.

We considered two classes, ω_1 and ω_2, which are represented by two normal random vectors, $\mathbf{X}_1 \sim N(M_1, \Sigma_1)$ and $\mathbf{X}_2 \sim N(M_2, \Sigma_2)$, respectively. For each class, we used two sets of 100 normal random points to test the accuracy of the classifiers.

In all the cases, to display the distribution, we plotted the ellipsoid of equi-probable points instead of the training points. This was because the plot of the three dimensional points caused too much cluttering, making the shape of the classes and the discriminants indistinguishable.

8.4.1 Linear Discriminants for Two Diagonalized Classes

In the first test, DD-1, we considered the pairwise linear discriminant function for two diagonalized classes. These classes are normally distributed with covariance matrices

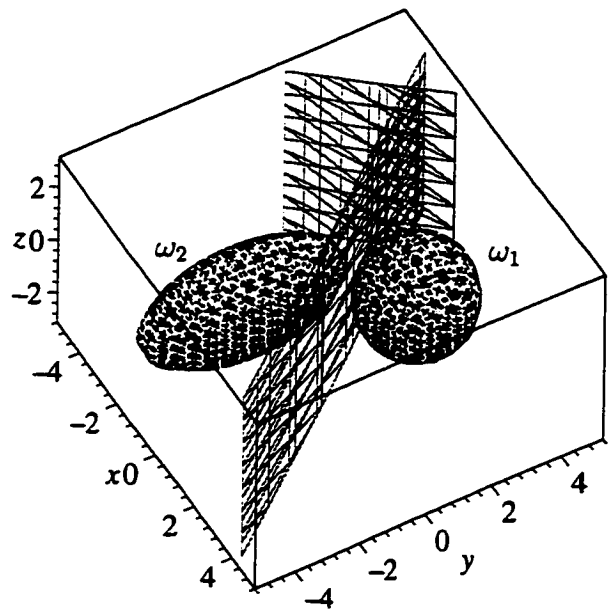


Figure 8.1: Example of pairwise linear discriminant for diagonalized normally distributed classes. This example corresponds to the data set DD-1.

being the identity matrix and another matrix in which two elements of the diagonal are not equal to unity and the remaining are unity. This is indeed, the case in which the optimal Bayes classifier is shown to be a pair of hyperplanes, stated and proven in Theorem 8.3. The following mean vectors and covariance matrices were estimated from 100 training samples to yield the respective classifier:

$$\textbf{DD-1: } M_1 = -M_2 \approx \begin{bmatrix} 1.037 \\ 2.049 \\ 0 \end{bmatrix}, \Sigma_1 = I, \Sigma_2 \approx \begin{bmatrix} .481 & 0 & 0 \\ 0 & 3.131 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The plot of the ellipsoids simulating the points and the linear discriminant hyperplanes in the three dimensional space are depicted in Figure 8.1. The accuracy of the classifier was 96% for ω_1 and 97% for ω_2. The power of the scheme is obvious!

8.4.2 Pairwise Linear Discriminant with Different Means

To demonstrate the properties of the classifier satisfying the conditions of Theorem 8.4, we considered the pairwise linear discriminant with different means. In this case, the

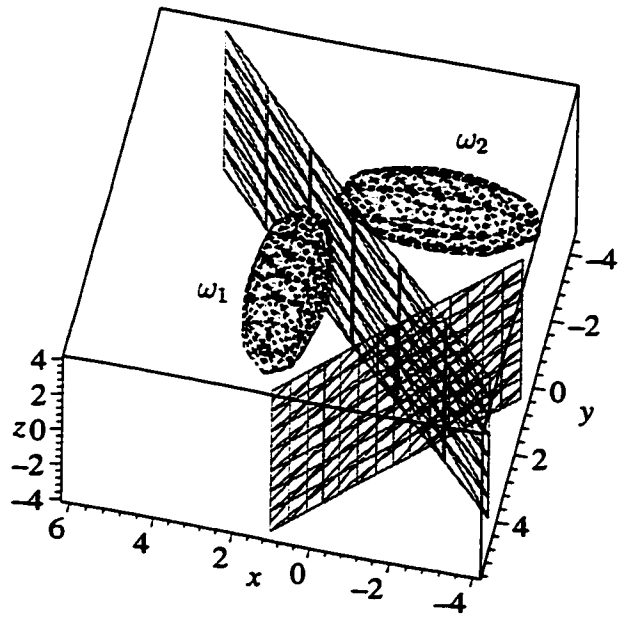


Figure 8.2: Example of pairwise linear discriminant with different means for the case described in Section 8.2. These classes corresponds to the data set DM-1.

diagonal covariance matrices differ only in two elements. These two elements in the first matrix have switched positions in the second covariance matrix. The remaining elements are identical in both covariance matrices. The mean vectors and covariance matrices estimated from 100 training samples are given below.

$$\textbf{DM-1:}\ M_1 = -M_2 \approx \begin{bmatrix} 1.542 \\ 1.542 \\ 0 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} .384 & 0 & 0 \\ 0 & 2.121 & 0 \\ 0 & 0 & .475 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} 2.121 & 0 & 0 \\ 0 & .384 & 0 \\ 0 & 0 & .475 \end{bmatrix}$$

Using these parameters, the pairwise linear classifier was derived. The plot of the ellipsoids simulating the points and the linear discriminant hyperplanes are shown in Figure 8.2. With this classifier, we obtained an accuracy of 94% for ω_1 and 97% for ω_2.

8.4.3 Pairwise Linear Discriminant with Equal Means

We also tested our scheme for the case of the pairwise linear classifier with equal means, EM-1, for the generalized multi-dimensional Minsky's Paradox. This is the

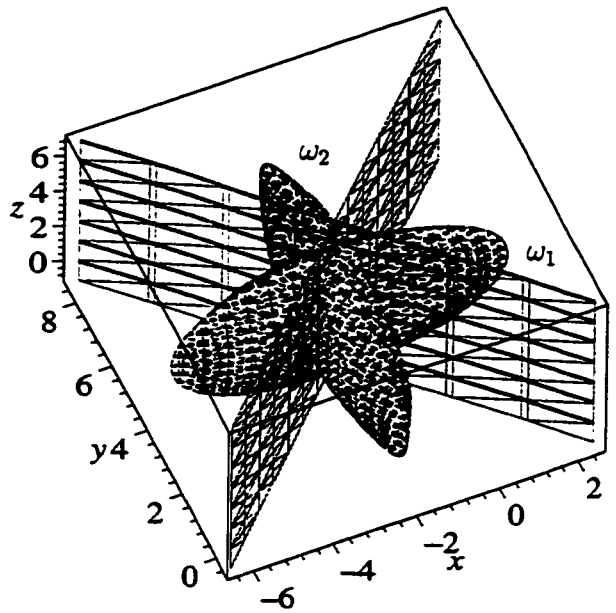


Figure 8.3: Example of pairwise linear discriminant with equal means for the case described in Section 8.3. The data set is EM-1. This resolves the generalized multi-dimensional Minsky's paradox.

case in which the distributions have coincident mean vectors, but covariance matrices as in the the case of DM-1. Two classes having parameters like these are proven in Theorem 8.5 to be optimally classified by a pair of hyperplanes. We obtained the following estimated mean vectors and covariance matrices from 100 training samples:

$$\textbf{EM-1:}\ M_1 = M_2 \approx \begin{bmatrix} -1.95 \\ 4.067 \\ 1.988 \end{bmatrix}, \Sigma_1 \approx \begin{bmatrix} 5.327 & 0 & 0 \\ 0 & .171 & 0 \\ 0 & 0 & .238 \end{bmatrix}, \Sigma_2 \approx \begin{bmatrix} .171 & 0 & 0 \\ 0 & 5.327 & 0 \\ 0 & 0 & .238 \end{bmatrix}$$

The shape of the overlapping classes and the linear discriminant function from these estimates are given in Figure 8.3. We evaluated the classifier with 100 randomly generated test points, and the accuracy was 82% for ω_1 and 85% for ω_2. Observe that such a linear classifier is not possible using any of the reported traditional methods.

Example	Accuracy for ω_1	Accuracy for ω_2
DD-1	96 %	97 %
DM-1	94 %	97 %
EM-1	83 %	88 %

Table 8.1: Accuracy of classification of 100 three dimensional random test points generated with the parameters of the examples presented above. The accuracy is given in percentage of points correctly classified.

8.4.4 Analysis of Accuracy

Finally, we analyze the accuracy of the classifiers for the different cases discussed above. The accuracy of classification for the three cases is given in Table 8.1. The first column corresponds to the test case. The second and third columns represent the percentage of correctly classified points belonging to ω_1 and ω_2, respectively. Observe that the accuracy of DD-1 is very high. This case corresponds to the pairwise linear discriminant when dealing with covariance matrices being the identity and another diagonal matrix in which two elements are not equal to unity, as shown in Theorem 8.3. The accuracy of the case in which the means are different and the covariance matrices as given in (8.5) and (8.6) (third row) is still very high. The fourth row corresponds to the case where the means are identical, referred to as EM-1. The accuracy is lower than that of the other cases but still very high, even though the classes overlap and the discriminant function is pairwise linear. This demonstrates the power of our scheme to resolve Minsky's Paradox in three dimensions !

8.5 Conclusions

In this chapter, we have extended the theoretical framework of obtaining optimal pairwise linear classifiers for normally distributed classes. We have shown that it is still possible to find the optimal classifier as a pair of hyperplanes for more than two dimensions even though the underlying covariance matrices are not equal.

To initiate this study, we first determined the necessary and sufficient conditions for obtaining an optimal pairwise linear classifier when the covariance matrices are the identity and a diagonal matrix. In this case, we have formally shown that it is

possible to find the optimal linear classifier by satisfying certain conditions specified in the planar projections of the various components.

In the second case, we have dealt with normally distributed classes having different mean vectors and with some special forms of covariance matrices. When the covariance matrices differ only in two elements of the diagonal, and these elements are permuted in their positions in the *second* covariance matrix, we showed that the optimal classifier is a pair of hyperplanes only if the mean vectors differ in the two elements of these positions. The conditions for this have been formalized too.

The last case that we have considered is the generalized Minsky's paradox for multi-dimensional normally distributed random vectors. By a formal procedure, we have found that when the classes are overlapping and the mean vectors are coincident, under certain conditions on the covariance matrices, the optimal classifier is a pair of hyperplanes. This resolves the multi-dimensional Minsky's paradox in its generality (i.e. where the features are not just points but normally distributed random vectors), open since 1957.

We have also provided some empirical examples for each of the cases discussed in this chapter, and we have tested our classifier on various three-dimensional normally distributed features. The classification accuracy obtained is very high, which is reasonable as the classifier is optimal in the Bayesian context. The degree of accuracy for the third case is not as high as that of the other cases, but is still impressive given the fact that we are dealing with significantly overlapping classes.

Chapter 9

A Formal Analysis of Why Heuristic Functions Work

The area of computer science has still quite a few open, unsolved problems. In this research work, we are concerned with one such problems, namely that of using *heuristics* to solve optimization problems.

Any arbitrary optimization problem[1] is typically defined in terms of instances which are drawn from a (finite) set, $\mathcal{X}$, an objective function, and some feasibility functions. The aim is to find an (and hopefully, the unique) instance of $\mathcal{X}$, which leads to the maximum (or the minimum) value of the objective function subject to the feasibility constraints. A formal definition of an optimization problem can be found in [77]. But to be more specific, consider the well-known Traveling Salesman Problem (TSP), in which the cities are numbered from 1 to n, and the salesman starts from city 1, visits every other city once, and returns to city 1. An instance of $\mathcal{X}$ is a permutation of the cities, for example, 1 4 3 2 5, if we are considering a world consisting of five cities. The objective function for that instance, $f(1\,4\,3\,2\,5)$ is obtained by performing the summation of the inter-city distances: $1 \rightarrow 4$, $4 \rightarrow 3$, $3 \rightarrow 2$, $2 \rightarrow 5$, and $5 \rightarrow 1$. The optimal solution is the instance that minimizes the value of f.

A *heuristic algorithm* is an algorithm that attempts to find a certain instance of $\mathcal{X}$ that maximizes f (or the *profit*) by iteratively invoking a *heuristic* function. The

[1]Every optimization problem can also be formulated as a decision problem [49].

instance that maximizes f will be the *optimal solution*[2] to the optimization problem. A *heuristic* is a method that performs one or more modifications to a given solution or instance, in order to obtain a different solution which is either superior, or which leads to a superior solution. The heuristic, in turn, invokes a heuristic function, which estimates (or measures) the cost of the solution at the particular state in the search process. This is the context in which we use these terms.

Many heuristic algorithms and heuristic functions have been reported in the literature, where the former include the *alpha-beta search* [86], *backtracking*, *hill-climbing* [77], *simulated annealing* [1], *genetic algorithms* [95], *tabu search* [52], *learning automata* [102], etc. The issue of how heuristic functions are used in such heuristic algorithms in searching, game playing, etc., can be found in [104, 124] and is, indeed, an enormous field of study in itself. This question is not addressed here.

To clarify issues, let us consider the classical n-puzzle problem [104]. This problem consists of a square board containing n square tiles and an empty position called the "blank". The aim is to rearrange the tiles from some pre-defined (usually random) initial configuration into a pre-determined goal configuration, by sliding any tile adjacent to the blank into the blank position. A heuristic algorithm solves this problem by examining, using a heuristic function, *some* of the possible *valid* movements. Viewed from the perspective of the underlying state graph, the possible states encountered at the next level form the children nodes of the current node in the search structure. Other variants of heuristic algorithms involve the examination of lower levels as well. The breadth-first search and depth-first search schemes are examples of heuristic algorithms, useful in any such problem solving strategy. An example of a heuristic function, however, is the measurement (or estimate) of the number of tiles that are out of place. Another heuristic is the sum of the depth of the node and the number of tiles that are out of place.

One of the better-known solutions to the n-puzzle problem is the A* algorithm. This algorithm is a graph search algorithm that is used to find the path of minimum cost between two nodes, the start node and the goal node. The A* maintains a tree

[2]We use the term "solution" to refer to an element $x \in \mathcal{X}$, and the term "profit" to refer to the value of $f(x)$. In minimization problems, $f(.)$ will be a cost funtion.

which stores the paths that are already explored. Using these paths, a measure, f, of the potential advantage of choosing each path is calculated. The value of f, which is the cost of traversing the graph between two nodes, can be calculated by using different heuristic functions. A heuristic is said to be *admissible*, and the A* converges to the correct result, if the heuristic function is an upper bound of the true cost from all nodes to the goal node.

In general, for any arbitrary problem, the question of how useful a heuristic function is, in determining the cost of traversing from one node to another, has no known analytic solution – it has traditionally been empirically analyzed. In this chapter, we present a formal analysis that provides a *stochastically positive* answer to the question of comparing the relative advantages of potential heuristic functions.

The A* algorithm and its variants (like the A+ algorithm) have also been successfully applied to other problems, such as object recognition using deformable templates [104, 140, 172]. Various solutions to optimization problems using different heuristic functions are found in [172]; we shall use this paper, [172], to highlight the difference between the heuristic algorithms, and the effect of the *same* algorithm using various potential heuristic functions. The authors of [172] address the problem of tracking roads in satellite images using the *twenty-question* search paradigm, and the A+ algorithm, a "cousin" of the A* algorithm. Using these algorithms the roads can be represented in terms of straight-line segments. The various paths are expanded by the application of an ensemble of heuristic functions. One such heuristic function is the one based on the conditional entropy measurements of the branches, which are used to choose the most "promising" path. While the paper discusses other heuristic functions, the question of how one can compare the solutions obtained using the various heuristic functions is achieved by comparing the empirical simulation results. We hope that our formal analysis can be a tool to achieve a more rigorous comparison of these heuristic functions in [172], and other similar scenarios[3].

The tools we propose to use utilize PR techniques. *Although we are aware of the use of PR principles in real life scenarios, we are not aware of any previous results in which*

[3]The model presented here has some limitations when investigating the quality of solutions yielded by an A*-like algorithm. These limitations will be discussed in a later sub-section.

PR principles have been used to solve a theoretical unsolved problem in a completely different field.

Our result can be crystallized as follows: Given two heuristic functions, the question of determining which is superior, has typically demanded a yes/no answer which is often substantiated based on empirical evidence. We have solved the problem of deciding on the *superior* heuristic function by using PR techniques. It should be mentioned that there are numerous well-known techniques that have been utilized in the context of pattern classification, such as hypothesis testing, bootstrap methods, Neyman-Pearson methods, etc. A good reference for such methods can be found in [115]. However, the results derived in this research work essentially use the methods that have been traditionally applied to optimal Bayesian Classification, as described in Chapter 6. Using these principles, we prove the following assertion: Given two heuristic functions, H_1 and H_2, used by a heuristic algorithm in finding a solution to a particular problem, if the accuracy in obtaining the *optimal* solution by using H_1 is greater than that of using H_2, then H_1 has a higher probability of leading to the optimal solution than H_2. To the best of our knowledge, this is an open problem. *However, this unproven conjecture has been the basis for designing numerous algorithms such as the A^* algorithm, and its variants, in searching, game playing, and numerous other applications* [104, 124, 128, 172].

Our strategy for achieving this analysis is as follows. The first task is to model the cost of the solution. Since the optimal "true" cost is unknown, we represent *it* in terms of its estimate, as estimated using the heuristic function. Observe that since the latter is inaccurate, this "cost" is represented in terms of a random variable. Note that by "cost", we do not mean the cost of the search process involved in determining the optimal solution, but rather the cost of the optimal solution, as estimated by the heuristic function. This difference is crucial.

Now that the modeling of the heuristic function is in place, the question of quantifying the quality of any heuristic function has to be considered. Informally speaking, we can say that this research work concerns this "heuristic-function quality assessment" problem, which is addressed, in turn, by viewing it as a pattern recognition problem. We solve *this* pattern recognition problem by considering two independent random vari-

ables, the first for the optimal solution and the second for the sub-optimal one, both of them being pursued by a heuristic function, H_1. We use a reasonable model for the accuracy of the heuristic function, in which the error of H_1 is a doubly-exponential random variable[4]. This distribution, which as we shall presently see, is used to approximate the Gaussian distribution, is typically used in reliability and failure models, and hence is reasonable in this scenario. In our model, the accuracy of the heuristic function is related to the variance of the random variable used to represent it. The analysis for the Gaussian distribution follows thereafter.

If we now consider another heuristic function, H_2, whose variance is greater than that of H_1, and whose mean is the same as that of H_1, we have a model by which the efficiency of heuristic functions can be compared. Indeed, using *this* model, we have theoretically proven that H_1 is more likely to succeed in obtaining the optimal solution than H_2. For this model, we have also proved the uniqueness of the result, and the conditions for which both heuristic functions lead to coincident probabilities of success.

The doubly exponential distribution is actually meant to be an approximation of the Gaussian distribution, typically used to model errors. However, the algebraic analysis for Gaussian distributions is impossible as there is no closed-form expression for integrating its probability density function. Consequently, we have extended the analysis for the doubly exponential distribution to formulate a reasonable analysis for the Gaussian distribution using numerical integration. By means of this analysis, we have corroborated the validity of our hypothesis for Gaussian distributions also.

We also provide empirical results on using a few histogram-like estimation methods in database query optimization, which demonstrate the validity of our theoretical analysis.

The results presented in this chapter have also been published in [105, 106, 108, 112].

[4]The reasoning used in this research work assumes that the errors are on either side of the true value. However, we believe that if the distribution is one-sided, similar arguments will be true as long as the distribution is not "heavily-tailed".

9.1 Problem Motivation

9.1.1 Applications

There are many heuristic algorithms that can be used to solve a wide variety of NP-hard problems. Such problems can be found in a wide range of applications spanning the whole spectrum of artificial intelligence, and include game playing and game theory, graph theory, database query optimization, networking, computational geometry, number theoretic problems, parallel processing, etc. The results presented in this chapter are applicable to any heuristic algorithm that uses different heuristic functions to solve a particular problem. In this introductory section, we just describe a few of them.

In the area of database query optimization, when more than two tables have to be joined, intermediate join operations are performed to ultimately obtain the final relation. As a result, the same query can be performed by means of different intermediate (join) operations. A simple sequence of join operations that leads to the same final result is called a *query evaluation plan* (QEP). Each QEP has associated an internal cost, which depends on the number of operations performed in the intermediate joins. The problem of choosing the best QEP is a combinatorially explosive optimization problem. This problem is currently solved by estimating the query result sizes of the intermediate relations and selecting the most efficient QEP.

Since the analysis of selecting the best QEP must be done in "real" time, it is not possible to inspect the real data in this phase. Consequently, query result sizes are usually estimated using statistical information about the structures and the data maintained in the database catalogue. This information is used to approximate the distribution of the attribute values in a particular relation. Hence the problem of selecting the best QEP depends on how well that distribution is approximated.

In [68], it has been shown that errors in query result size estimates may increase exponentially with the number of joins. Since current databases and the associated queries increase in complexity, numerous efforts have being made to devise more efficient techniques that solve the query optimization problem.

Many techniques have been proposed to estimate query result sizes, including histograms, sampling, and parametric techniques [76, 91, 100, 116]. Histograms are the

most commonly used form of statistical information. They are incorporated in most of the commercial database systems such as Oracle, Microsoft SQL Server, Teradata, and DB2, which mainly use the Equi-depth histogram. The prominent models of histograms known in the literature are: *Equi-width* [24, 76], *Equi-depth* [100, 116], the *Rectangular Attribute Cardinality Map* (*R-ACM*) [109], the *Trapezoidal Attribute Cardinality Map* (*T-ACM*) [110], and the *V-Optimal Histograms* [68, 118].

In this scenario, the heuristic algorithm is the actual algorithm that uses a histogram as the heuristic function, and obtains an optimal (or a sub-optimal) QEP. The heuristic function used by this algorithm is the *actual histogram* that approximates the distribution of the attribute values of the relevant tables. Thus, in our model (and using our terminology), Equi-width, Equi-depth, the R-ACM and the T-ACM are the heuristic functions.

Other areas in which our model can be used to answer open questions are in the fields of *game theory* and *game playing* [128]. In game playing, the most widely used structure used to analyze the best possible move and strategy is a *game tree*, whose root node represents the initial status of the board. All possible moves of the first player are the edges from the root to the first level, the edges of each child represent all possible moves of the second player, the opponent. Continuing in the same fashion, the game is played (or rather plans executed) until one of the players wins. The aim is to optimize the moves of the first player based on searching *all* the branches of the tree until the leaves, and perform the best move based on maximizing the reward of the first player and minimizing that of the second one.

There are many techniques used to optimize the moves of the first the player. One of them is the *minimax search algorithm*, which searches over a fixed number of levels of the entire tree, and finds the best moves at each node. This exhaustive searching procedure has a complexity that grows exponentially with the number of nodes of the tree. A more efficient mechanism is the *alpha-beta search* algorithm [86], a heuristic that significantly reduces the number of nodes explored. Both of these assume that the heuristic function that they use, which typically evaluates the position of the board viewed from the perspective of the first player, is advantageous in determining a superior strategy. This is the question that we address in this chapter. The model

presented in this chapter has important consequences in choosing such a heuristic function. Such a heuristic function could be, for example, the cost of a path from the current state to a goal state, which unfortunately is not exactly known, but is estimated. The searching scheme, such as the alpha-beta search and the minimax search algorithm, uses this heuristic function to search for a, hopefully, *optimal* path in the game tree.

Another application of our result is in graph theory, for example, in solving the uniform graph partitioning problem. Given a complete graph on $2n$ vertices, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, along with a cost function $f : \mathcal{E} \rightarrow \mathbb{Z}^+ \bigcup \{0\}$, the aim is to find a partition whose sum of costs of the individual subsets is minimized. This problem is also known to be NP-hard, and has several applications especially in VLSI design, hydrology, networks, etc. Many heuristic algorithms have been proposed to solve this problem, including simulated annealing, genetic algorithms, learning automata, etc. [77, 111]. When considering a particular heuristic algorithm, we can incorporate different heuristic functions to approximate the sum of costs of the individual subsets of a particular partitioning. It is intuitive that a more accurate heuristic function is more likely to succeed in finding the optimal solution. However, this is not what happens in all cases. We rather provide a stochastic answer to this question. By means of a rigorous theoretical analysis, we prove that a particular heuristic function, which provides more accurate approximations for the sum of costs of the individual subsets, is more likely to obtain the minimal cost for a partitioning, than a *less accurate* heuristic function.

9.1.2 Problem Statement

In this chapter, we propose a theoretical model that solves this fundamental open problem in computer science, namely that of relating heuristic functions with solution optimality, using the principles of the theory of *pattern classification*. This problem has been (to our knowledge) open. In particular, the corresponding database query optimization problem has been unsolved for more than two decades.

More specifically, we prove the following: Given a heuristic algorithm, A, that invokes two heuristic functions, H_1 and H_2, used in a decision problem, if the accuracy

in approximating the optimal solution by using H_1 is greater than that of using H_2, then H_1 has a higher probability of leading to the optimal solution than H_2.

The importance of the results of this chapter is that we show that the answer to the accuracy/optimality question is "stochastically positive". In other words, we prove that although a superior heuristic function may not always yield a better solution, the probability that the superior heuristic function yields an optimal solution exceeds the probability that an inferior heuristic function yields an optimal solution. This research work thus justifies and gives a formal rigorous basis for why heuristic functions work.

We analytically prove that under the well-acclaimed models of inaccuracy, the better the accuracy of a heuristic function, the greater the probability of it choosing the optimal solution. We have also provided some empirical results related to the field of database query optimization. These results show the superiority of the R-ACM over the traditional histogram estimation methods, the Equi-width and the Equi-depth. The empirical results obtained by testing these properties for many of the above histogram methods in random databases show that the R-ACM is significantly superior to both the Equi-width and the Equi-depth schemes.

9.1.3 Restrictions of Our Model

As mentioned above, this chapter addresses the problem of quantifying the quality of a heuristic function, and it achieves this by posing the problem in a fairly general framework. However, for the results to be applicable for a particular application domain which uses a specific search strategy such as the A^* algorithm, the logistics of the search process itself will have to be considered.

Informally speaking, our main result proves the following: Given two heuristic functions evaluating the same "cost", a search mechanism utilizing these functions will converge (with a higher probability) to a superior solution, when it utilizes a function with a lesser variance. However, comparing the performance of heuristic functions in the search process initiated by A^* is a more complicated issue. The reason for this can be argued as follows. In each iteration, A^* computes the values of the heuristic function (say, "$f(.)$") for all candidate nodes (the OPEN list), which represent how

promising they are. A^* then selects the one with the highest value of $f(.)$, generates *its* children, computes *their* values of $f(.)$, and inserts them into the OPEN list. For an algorithm like A^*, the most we can claim is that it is more expedient to use a heuristic function which better estimates the "cost", than one which estimates it poorly. The question of how the nodes in the OPEN list lead to solutions, is really a problem-dependent question which we cannot answer here. We intend to study this problem in the database query optimization domain mentioned later, by incorporating a *search* strategy to search the set of QEPs whose costs are estimated by the various histogram methods. Note that this does not invalidate the query-optimization results presented in this chapter, because, in our simulations, we exhaustively search the QEP space without using any intelligent search strategy like A^*.

9.2 Heuristic Functions : Accuracy vs. Optimality

Consider a heuristic algorithm, A, that invokes either of two heuristic functions, H_1 and H_2. The probability of correctly estimating a cost value of a particular solution by H_1 and that of estimating a cost value by H_2 are represented by two independent random variables. In our model, we assume that these two heuristic functions are independent, and thus, the value obtained by one heuristic function should not affect the value obtained by the second.

For the analysis done below, we work with two models for the error function: the doubly exponential distribution and the normal distribution. In the former, the probability of obtaining a value that deviates from the mean (or true value) falls exponentially as a function of the deviation. The exponential distribution is more typical in reliability analysis and in failure models, and in this particular domain, the question is one of evaluating how reliable the quality of a solution is if only an estimate of its performance is available. More importantly, it is used as an approximation to the Gaussian distribution for reasons which will be clarified momentarily. The Gaussian model is much more difficult to analyze, since there is no closed-form algebraic expression for integrating the probability density function. However, a formal computational proof is included, which confirms our hypothesis.

9.2.1 Analysis Using Exponential Distributions

A random variable, X, is said to be *doubly exponentially* distributed with parameter λ if the density function is given by:

$$f_X(x) = \frac{1}{2}\lambda e^{-\lambda|x-c|} \qquad -\infty < x < \infty \ . \tag{9.1}$$

If X is a doubly exponential random variable, by elementary integration and straightforward algebraic steps, it can be shown that:

$$\mathrm{E}[X] = c \ , \text{ and} \tag{9.2}$$

$$\mathrm{Var}[X] = \frac{2}{\lambda^2} \ . \tag{9.3}$$

Without loss of generality, if the mean of the cost of the optimal solution is c_1, by shifting the origin by c_1, we can work with the assumption that the cost of the best solution is 0, which is the mean of these two random variables. The cost of the second best solution is given by another two random variables (one for H_1 and the other one for H_2) whose mean, $c_2 > 0$, is the same for both variables. An example will help to clarify this.

Example 9.1. Suppose that using H_1 leads to the optimal cost with a probability represented by a doubly exponential random variable, $X_1^{(opt)}$, whose mean is 0 and $\lambda_1 = 0.4$. This heuristic function also leads to another sub-optimal cost according to $X_1^{(subopt)}$ whose mean is 8 and $\lambda_1 = 0.4$.

H_2 is another heuristic function using which the optimal cost is chosen with a probability distribution given by $X_2^{(opt)}$ whose parameters are $c_1 = 0$ and $\lambda_2 = 0.2$. It leads to the second sub-optimal cost value with a probability density given by $X_2^{(subopt)}$ whose parameters are $c_2 = 8$ and $\lambda_2 = 0.2$.

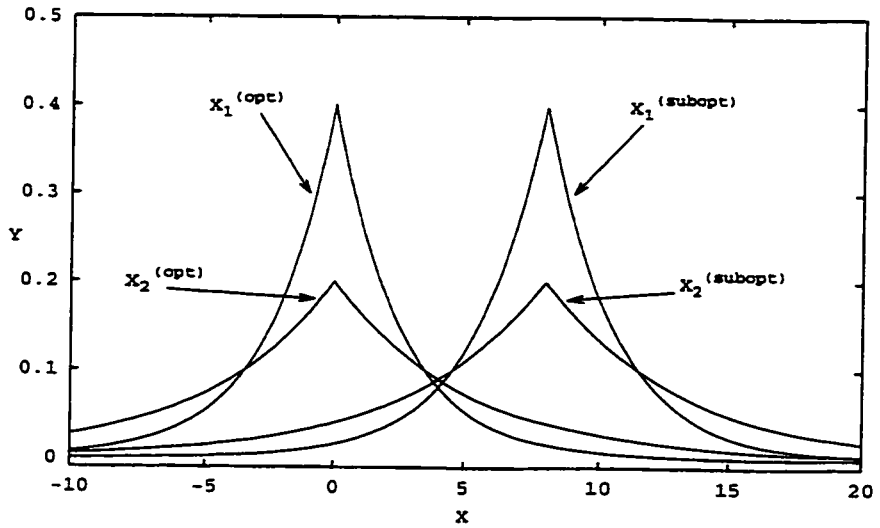


Figure 9.1: An example of doubly exponential distributions for the random variables $X_1^{(opt)}$, $X_2^{(opt)}$, $X_1^{(subopt)}$ and $X_2^{(subopt)}$, whose parameters are $\lambda_1 = 0.4$ and $\lambda_2 = 0.2$.

The fact that $\frac{2}{\lambda_1^2} < \frac{2}{\lambda_2^2}$ signifies that the probability of using H_1 could lead to a sub-optimal cost is smaller than the probability of using H_2 leading to a sub-optimal cost. This scenario is depicted in Figure 9.1, and is formalized presently. □

The result depicted above is formalized in the following theorem, *which is the first primary result of this chapter, and answers the open question referred to above.* The theorem is formulated in terms of the probabilities that the two heuristic functions lead to the wrong decision, which we show is inherently related to the probability that these heuristic functions lead to the convergence to the *sub-optimal* solutions. The formulation of the result and the proof utilize techniques typically foreign to database theory, game theory, artificial intelligence, or for that matter any computer science area in which this approach can be applied. They belong to the theory of PR.

The second theorem, extends the results of the first, and shows how the results can also be geometrically interpreted.

Theorem 9.1. Suppose that A is a heuristic algorithm that can potentially utilize either of two heuristic functions, H_1 and H_2. Let:

- X_1 and X_2 be two doubly exponential random variables that represent the esti-

mated costs of the *optimal* solutions obtained by using H_1 and H_2 respectively.

- X_1' and X_2' be two other doubly exponential random variables representing the estimated costs of *non-optimal* solutions obtained by using H_1 and H_2 respectively.
- $0 = \mathrm{E}[X_1] = \mathrm{E}[X_2] \leq \mathrm{E}[X_1'] = \mathrm{E}[X_2'] = c$.
- p_1 and p_2 be the probabilities that H_1 and H_2 respectively lead to the wrong decision.

Then,

$$\text{if } \mathrm{Var}[X_1] = \mathrm{Var}[X_1'] = \frac{2}{\lambda_1^2} \leq \frac{2}{\lambda_2^2} = \mathrm{Var}[X_2] = \mathrm{Var}[X_2'], \quad p_1 \leq p_2 \ .$$

Proof. Consider a particular cost, x. The probability that x leads to a wrong decision when A uses H_1 is that of incorrectly classifying x as being obtained from the non-optimal solution. This is, indeed, the error in classification, and is the area under the curve of the pdf function of X_1' or the cumulative probability of x under the pdf of H_1 when it refers to the sub-optimal solution. Because of the discontinuity of the doubly exponential function at c, this area is decomposed into the following two integrals:

$$\begin{aligned} I_{11} &= \textstyle\int_{-\infty}^{x} \frac{1}{2}\lambda_1 e^{\lambda_1(u-c)}\, du && \text{if } x < c \text{ , and} \\ I_{12} &= \textstyle\int_{-\infty}^{c} \frac{1}{2}\lambda_1 e^{\lambda_1(u-c)}\, du + \int_{c}^{x} \frac{1}{2}\lambda_1 e^{-\lambda_1(u-c)}\, du && \text{if } x > c \ . \end{aligned} \tag{9.4}$$

Solving the integrals, (9.4) results in:

$$\begin{aligned} I_{11} &= \tfrac{1}{2}e^{\lambda_1(x-c)} - \lim_{u\to-\infty} \tfrac{1}{2}e^{-\lambda_1(u-c)} && = \tfrac{1}{2}e^{-\lambda_1(x-c)} \text{ , and} \\ I_{12} &= \lim_{u\to-\infty} \tfrac{1}{2}e^{-\lambda_1(-u+c)} + \tfrac{1}{2} - \tfrac{1}{2}e^{-\lambda_1(x-c)} + \tfrac{1}{2} && = 1 - \tfrac{1}{2}e^{-\lambda_1(x-c)} \ . \end{aligned} \tag{9.5}$$

The probability that using H_1 leads to the wrong decision for *all* the values of x is the following function of λ_1 and c:

$$p_1 = I(\lambda_1, c) = \int_{-\infty}^{0} I_{11}\frac{1}{2}\lambda_1 e^{\lambda_1 x}\,dx + \int_{0}^{c} I_{11}\frac{1}{2}\lambda_1 e^{-\lambda_1 x}\,dx + \int_{c}^{\infty} I_{12}\frac{1}{2}\lambda_1 e^{-\lambda_1 x}\,dx\ . \quad (9.6)$$

which, after applying the distributive law and substituting the values of I_{11} and I_{12}, can be written as:

$$\int_{-\infty}^{0} \frac{\lambda_1}{4} e^{2\lambda_1 x - \lambda_1 c}\,dx - \int_{0}^{c} \frac{\lambda_1}{4} e^{-\lambda_1 c}\,dx + \int_{c}^{\infty} \left[\frac{\lambda_1}{2} e^{-\lambda_1 x} - \frac{\lambda_1}{4} e^{-2\lambda_1 x + \lambda_1 c}\right] dx\ . \quad (9.7)$$

After solving the integrals, (9.7) is transformed into:

$$\frac{1}{8}e^{-\lambda_1 c} + \frac{1}{4}\lambda_1 c e^{-\lambda_1 c} + \frac{3}{8}e^{-\lambda_1 c} = \frac{1}{2}e^{-\lambda_1 c} + \frac{1}{4}\lambda_1 c e^{-\lambda_1 c}\ . \quad (9.8)$$

Similarly, we do the same analysis for p_2, which is a function of λ_2 and c:

$$p_2 = I(\lambda_2, c) = \frac{1}{2}e^{-\lambda_2 c} + \frac{1}{4}\lambda_2 c e^{-\lambda_2 c}\ . \quad (9.9)$$

We have to prove that:

$$p_1 = \frac{1}{2}e^{-\lambda_1 c} + \frac{1}{4}\lambda_1 c e^{-\lambda_1 c} \leq \frac{1}{2}e^{-\lambda_2 c} + \frac{1}{4}\lambda_2 c e^{-\lambda_2 c} = p_2\ . \quad (9.10)$$

Multiplying both sides by 2, and substituting $\lambda_1 c$ for α_1 and $\lambda_2 c$ for α_2, (9.10) can be written as follows:

$$e^{-\alpha_1} + \frac{1}{2}\alpha_1 e^{-\alpha_1} \leq e^{-\alpha_2} + \frac{1}{2}\alpha_2 e^{-\alpha_2}\ . \quad (9.11)$$

Substituting α_2 for $k\alpha_1$, $\alpha_1 \geq 0$ and $0 < k \leq 1$, (9.11) results in:

$$q_1 = e^{-\alpha_1} + \frac{1}{2}\alpha_1 e^{-\alpha_1} \leq e^{-k\alpha_1} + \frac{1}{2}k\alpha_1 e^{-k\alpha_1} = q_2 . \tag{9.12}$$

We now prove that $q_1 - q_2 \leq 0$. After applying natural logarithm to both sides of (9.12) and some algebraic manipulations, $q_1 - q_2 \leq 0$ implies:

$$F(\alpha_1, k) = k\alpha_1 - \alpha_1 + \ln(1 + \frac{1}{2}\alpha_1) - \ln(1 + \frac{1}{2}k\alpha_1) \leq 0 . \tag{9.13}$$

To prove that $F(\alpha_1, k) \leq 0$, we use the fact that $\ln x \leq x - 1$. Hence, we have:

$$\begin{aligned} F(\alpha_1, k) &= \alpha_1(k-1) + \ln\left(\frac{1 + \frac{1}{2}\alpha_1}{1 + \frac{1}{2}k\alpha_1}\right) && (9.14)\\ &\leq \alpha_1(k-1) + \frac{1 + \frac{1}{2}\alpha_1}{1 + \frac{1}{2}k\alpha_1} - 1 && (9.15)\\ &= \alpha_1(k-1) + \frac{\alpha_1 - k\alpha_1}{2 + k\alpha_1} && (9.16)\\ &= \frac{k\alpha_1 + k^2\alpha_1^2 - \alpha_1 - k\alpha_1^2}{2 + k\alpha_1} && (9.17)\\ &= \frac{\alpha_1(k-1)(k\alpha_1 + 1)}{2 + k\alpha_1} \leq 0 , && (9.18) \end{aligned}$$

because:

(*i*) $0 < k \leq 1$ and $\alpha_1 \geq 0 \Rightarrow \alpha_1(k-1) \leq 0$ and $k\alpha_1 + 1 > 0$. Hence $\alpha_1(k-1)(k\alpha_1 + 1) \leq 0$, and

(*ii*) $0 < k \leq 1$ and $\alpha_1 \geq 0 \Rightarrow 0 < k\alpha_1 \leq \alpha_1 \Rightarrow k\alpha_1 + 2 > 2 > 0$.

Hence the theorem. □

The above theorem can be viewed as a "sufficiency result". In other words, we have shown that $q_1 - q_2 \leq 0$ or that $p_1 \leq p_2$. We now show a "necessity result" stated as a

uniqueness result. This result states that the function $p_1 \leq p_2$ has its equality ONLY at the boundary condition where the two distributions are exactly identical.

To prove the necessity result, we consider $q_2 - q_1$ which, derived from (9.12), can be written, as a function of α_1 and k, as:

$$G(\alpha_1, k) = e^{-k\alpha_1} + \frac{1}{2}k\alpha_1 e^{-k\alpha_1} - e^{-\alpha_1} - \frac{1}{2}\alpha_1 e^{-\alpha_1} . \tag{9.19}$$

By examining its partial derivatives, we shall show that there are two solutions for equality. Furthermore, when $\alpha_1 \geq 0$ and $0 < k \leq 1$, we shall see that for a given k, there is only one solution, namely $\alpha_1 = 0$ and k, $0 < k \leq 1$, proving the uniqueness.

Theorem 9.2. Suppose that $\alpha_1 \geq 0$, $0 < k \leq 1$. Let $G(\alpha_1, k)$ be:

$$G(\alpha_1, k) = e^{-k\alpha_1} + \frac{1}{2}k\alpha_1 e^{-k\alpha_1} - e^{-\alpha_1} - \frac{1}{2}\alpha_1 e^{-\alpha_1} . \tag{9.20}$$

Then $G(\alpha_1, k) \geq 0$, and there are exactly two solutions for $G(\alpha_1, k) = 0$, being: $\{\alpha_1 = -1, k = 1\}$ and $\{\alpha_1 = 0, k\}$.

Proof. We must prove that, as defined in the theorem statement, $G(\alpha_1, k) \geq 0$.

We shall prove that this is satisfied by determining the local minima for $G(.,.)$, where $\alpha_1 \geq 0$ and $0 < k \leq 1$. We first find the partial derivatives of (9.20) with respect to α_1 and k:

$$\frac{\partial G}{\partial \alpha_1} = -\tfrac{1}{2}ke^{-k\alpha_1} - \tfrac{1}{2}k^2\alpha_1 e^{-k\alpha_1} + \tfrac{1}{2}e^{-\alpha_1} + \tfrac{1}{2}\alpha_1 e^{-\alpha_1} = 0 , \text{ and} \tag{9.21}$$

$$\frac{\partial G}{\partial k} = -\tfrac{1}{2}\alpha_1 e^{-k\alpha_1} - \tfrac{1}{2}k\alpha_1^2 e^{-k\alpha_1} = 0 . \tag{9.22}$$

We now solve (9.21) and (9.22) for α_1 and k. Equation (9.22) can be written as follows:

$$-\frac{1}{2}\alpha_1 e^{-k\alpha_1} = \frac{1}{2}k\alpha_1^2 e^{-k\alpha_1}, \tag{9.23}$$

which, after canceling some terms results in $k\alpha_1^2 + \alpha_1 = 0$. Solving this equation for α_1, we have: $\alpha_1 = -\frac{1}{k}$ and $\alpha_1 = 0$. Substituting $\alpha_1 = -\frac{1}{k}$ in (9.21), and canceling some terms, we obtain:

$$\frac{1}{2}e^{-\alpha_1} + \frac{1}{2}\alpha_1 e^{-\alpha_1} = 0, \tag{9.24}$$

which results in the solution to be $\alpha_1 = -1$, and consequently, $k = 1$.

The second root, $\alpha_1 = 0$, indicates that the minimum is achieved for any value of k.

We have thus found two solutions for (9.21) and (9.22), $\{\alpha_1 = 0, k\}$ and $\{\alpha_1 = -1, k = 1\}$. Since $\alpha_1 \geq 0$, it means that α_1 can have at least a value of 0, and hence the local minima is in $\{\alpha_1 = 0, k\}$. Substituting these two values in G, we see that $G(\alpha_1, k) = 0$, which is the minimum. Therefore, $G(\alpha_1, k) \geq 0$ for $\alpha_1 \geq 0$ and $0 < k \leq 1$.

Hence the theorem. □

To get a physical perspective of these results, let us analyze the geometric relation of the function G and the heuristic functions. G is a positive function in the region $\alpha_1 \geq 0$, $0 < k \leq 1$. When $\alpha_1 \to 0$, $G \to 0$. This means that for small values of α_1, G is also small. Since $\alpha_1 = \lambda_1 c$, the value of α_1 depends on λ_1 and c. When c is small, G is very close to its minimum, 0, and hence both probabilities, p_1 and p_2, are very close. This behavior can be noticed in Figure 9.2, and the phenomenon is observed if the heuristic functions are both comparable and almost equally efficient.

In terms of histogram methods and in database query optimization, when c is small, the optimal and the sub-optimal QEP are very close. Since histogram methods such as the Equi-width and the Equi-depth produce a larger error than the R-ACM and the T-ACM, the former are less likely to find the optimal QEP than the latter.

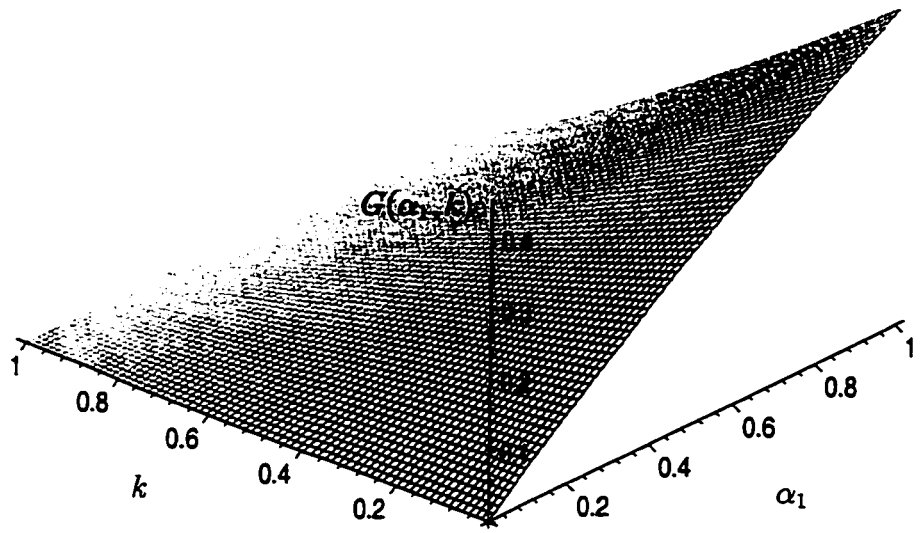


Figure 9.2: Function $G(\alpha_1, k)$ plotted in the ranges $0 \leq \alpha_1 \leq 1$ and $0 \leq k \leq 1$.

Interpreted alternatively, G is very small when λ_1 is close to 0. This means that $\text{Var}[X_1]$ is very large. Since $\text{Var}[X_1] \leq \text{Var}[X_2]$, $\text{Var}[X_2]$ is also very large, and both are close each other (In Figure 9.1, we would observe almost flat curves for both distributions). Random variables for histogram methods such as the Equi-width and the Equi-depth yield similar error estimation distributions with large and similar variances. Hence, the probabilities p_1 and p_2 are quite close, and consequently, similar results are expected for these estimation methods. However, when the heuristic functions yield widely different estimated costs (as in the case when the new histogram methods, the R-ACM and the T-ACM, are compared to the traditional methods), these effectively imply random variables with smaller variances being compared to random variables with larger variances. In such a case, the value of G is very high – implying that the former would yield superior solutions.

9.2.2 Analysis Considering Normal Distributions

For the analysis done in this section, we consider that we are given two heuristic functions, H_1 and H_2, for which the probabilities of choosing optimal or suboptimal solutions are represented by two normally distributed random variables, X_1 and X_2,

whose means are μ_1 and μ_2, and whose variances are σ_1^2 and σ_2^2 respectively.

Although the model using normal distributions is more realistic in real life problems, the analysis becomes impossible because there is no closed-form algebraic expression for integrating the normal probability density function. Alternatively, we have used numerical integration and we have obtained rather representative values for which the implication between efficiency and optimality is again corroborated.

Without loss of generality, if the mean cost of the optimal solution is μ_1, by shifting the origin by μ_1, we again assume that the cost of the best solution is 0, which is the mean of these two random variables. The cost of the second best solution is given by another two random variables (one for using the heuristic function H_1, and the other one for using the heuristic function H_2) whose mean, $\mu_2 > 0$, is the same for both variables. We also assume that, by scaling both distributions[5], the variance of using H_1 and leading to the optimal solution is unity. An example will help to clarify this.

Example 9.2. Suppose that using H_1 leads to the optimal cost with probability represented by the normal random variable $X_1^{(opt)}$ whose mean is 0 and standard deviation is $\sigma_1 = 1$. This heuristic function also estimates another sub-optimal cost according to $X_1^{(subopt)}$ whose mean is 4 and $\sigma_1 = 1$.

H_2 is another heuristic function that is used to estimate the optimal cost with probability given by $X_2^{(opt)}$ whose parameters are $\mu = 0$ and $\sigma_2 = 1.4$. The other corresponding sub-optimal cost given by the heuristic function H_2 is obtained with probability given by $X_2^{(subopt)}$ whose parameters are $\mu = 4$ and $\sigma_2 = 1.4$.

Observe that $\sigma_1 < \sigma_2$, and hence we are expecting that the probability of using H_1 and leading to a wrong decision is smaller than that of using H_2. The probability density functions for these four random variables are depicted in Figure 9.3. Note that, as in the doubly exponential distribution, given a particular value of x, if its probability under $X_1^{(opt)}$ is high, then the area for which using H_1 leads to the wrong decision (i.e. its cumulative probability under $X_1^{(subopt)}$) is small. Since these two quantities are multiplied and integrated, the final value is smaller than that of using H_2, since σ_2 is

[5]This can be done by multiplying σ_1^2 and σ_2^2 by σ_1^{-2}, and μ_1 and μ_2 by σ_1^{-1}. This is a particular case of the simultaneous diagonalization (discussed in Chapter 6) between d-dimensional normal random vectors for which $d = 1$.

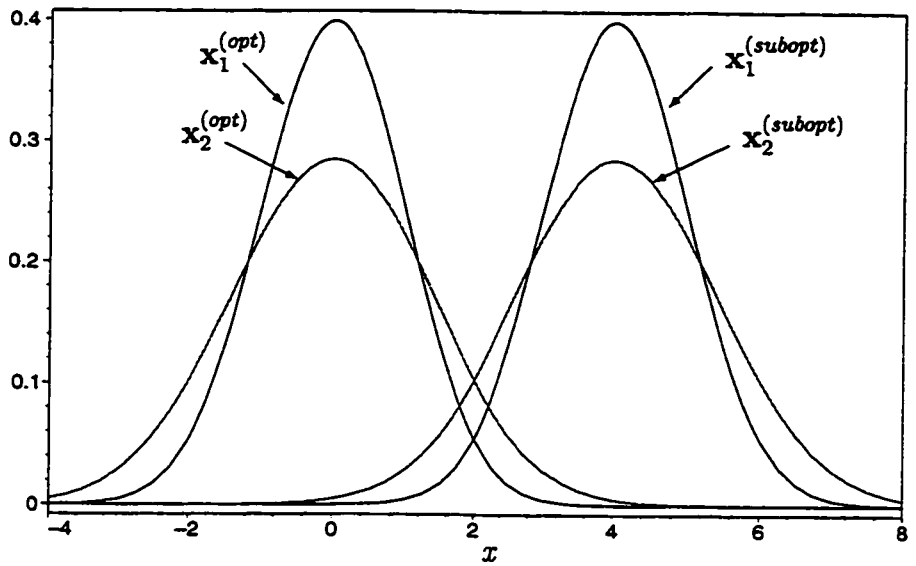


Figure 9.3: An example showing the probability density function of four normal random variables whose parameters are $\sigma_1 = 1$, $\sigma_2 = 1.4$, $\mu_1 = 0$, and $\mu_2 = 4$.

greater than $\sigma_1 = 1$. This is what we formally show below. □

Result 9.1. [6]

Suppose that A is a heuristic algorithm that can potentially utilize either of two heuristic functions, H_1 and H_2. Let:

- X_1 and X_2 be two normally distributed random variables that represent the costs of the *optimal* solutions obtained by H_1 and H_2 respectively.
- X_1' and X_2' be two other normally distributed random variables that represent the costs of *non-optimal* solutions obtained by using H_1 and H_2 respectively.
- $0 = \mathrm{E}[X_1] = \mathrm{E}[X_2] \leq \mathrm{E}[X_1'] = \mathrm{E}[X_2'] = \mu$.
- p_1 and p_2 be the probabilities that using H_1 and H_2 respectively lead to the wrong decision.

[6] We cannot claim this result as a theorem, since the formal analytic proof is impossible. This is because there is no closed-form expression for integrating the Gaussian probability density function. However, the computational proof that we present renders this to be more than a conjecture.

Then,

$$\text{if } \mathrm{Var}[X_1] = \mathrm{Var}[X_1'] = \sigma_1^2 \leq \sigma_2^2 = \mathrm{Var}[X_2] = \mathrm{Var}[X_2'], \quad p_1 \leq p_2 \ .$$

Computational Proof. To achieve this proof, we proceed by doing the same analysis that we did for the doubly exponential distributions (Theorem 9.1). If we consider a particular cost x, the probability that x leads to a wrong decision made by using H_1, is given by:

$$I_1 = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(u-\mu)^2}{2\sigma_1^2}} \, du \, . \tag{9.25}$$

The probability that using H_1 leads to the wrong decision for *all* the values of x is obtained by integrating the function resulting from multiplying every value of I_1 for each x with the respective probability density function of $X_1^{(opt)}$, which results in:

$$p_1 = \int_{-\infty}^{\infty} I_1 \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{x^2}{2\sigma_1^2}} \, dx \ . \tag{9.26}$$

Similarly, p_2 can also be expressed as follows:

$$p_2 = \int_{-\infty}^{\infty} I_2 \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{x^2}{2\sigma_2^2}} \, dx \ , \tag{9.27}$$

where I_2 is obtained in the same way as in (9.25) for the distribution with variance σ_2^2.

Since there is no closed-form algebraic expression for integrating the normal probability density function, no analytical solution for proving that $p_1 \leq p_2$ can be formalized.

Alternatively, we have invoked a computational analysis by calculating these integral for various representative values of σ_1 and σ_2 by using the trapezoidal rule. The values of $G = \frac{p_2}{p_1} \geq 1$ (i.e. for $1 \leq \sigma_1 \leq 10$ and $1 \leq \sigma_2 \leq 10$, where $\sigma_1 \leq \sigma_2$) are

$\sigma_1 \rightarrow$ σ_2 $\downarrow$	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00
1.00	1.00									
2.00	33.63	1.000								
3.00	73.92	2.198	1.000							
4.00	102.51	3.048	1.387	1.000						
5.00	122.20	3.634	1.653	1.192	1.000					
6.00	136.25	4.052	1.843	1.329	1.115	1.000				
7.00	146.61	4.360	1.983	1.430	1.200	1.076	1.000			
8.00	154.71	4.601	2.093	1.509	1.266	1.136	1.055	1.000		
9.00	161.05	4.789	2.179	1.571	1.318	1.182	1.098	1.041	1.000	
10.00	166.17	4.942	2.248	1.621	1.360	1.220	1.133	1.074	1.032	1.000

Table 9.1: Ratio between the probability of making the wrong decision for two normally distributed random variables whose standard deviations are σ_1 and σ_2.

depicted in Table 9.1 in the form of a *lower-diagonal* matrix. All the values of the *upper-diagonal* matrix (not shown here) are less than unity. Note that by making the value of $\sigma_1 = 1$, the analysis reduces to the first and second columns of this table. For example, if $\sigma_1 = 1$ and $\sigma_2 = 2$, $\frac{p_2}{p_1} \approx 33.6276$. For more neighboring values of σ_1 and σ_2, e.g. $\sigma_1 = 9$ and $\sigma_2 = 10$ ($\sigma_1 = 1$ and $\sigma_2 \approx 1.2345$ after scaling), $\frac{p_2}{p_1} \approx 1.0318$, which is very close to unity. The ratio for $\sigma_1 = 1$ and $\sigma_2 = 10$ is much bigger, i.e. more than one hundred times. □

In order to get a better perspective of the computational analysis, we study the behavior of the function $G = \frac{p_2}{p_1}$. Using the values of G given in Table 9.1, we have plotted this function in the three-dimensional space as $G(\sigma_1, \alpha_1)$, where $\alpha_1 = k\sigma_1$, $1 \leq k \leq 10$. The plot is depicted in Figure 9.4.

In order to enhance the visualization of G, we have approximated it by using the regression utilities of the symbolic mathematical software package Maple V [34]. When $k = 1$, the surface lies on the $z = 0$ plane, in the form of a straight line $x = y$ (labeled "$k = 1$ or $\sigma_1 = \sigma_2$" in the figure). This is the place in which G reaches its minimum, when both heuristic functions have identical variances. When k is larger (i.e. $k = 10$), the function G becomes much larger (up to 166 in Table 9.1). This clearly shows the importance of minimizing the variance in deciding on a heuristic function.

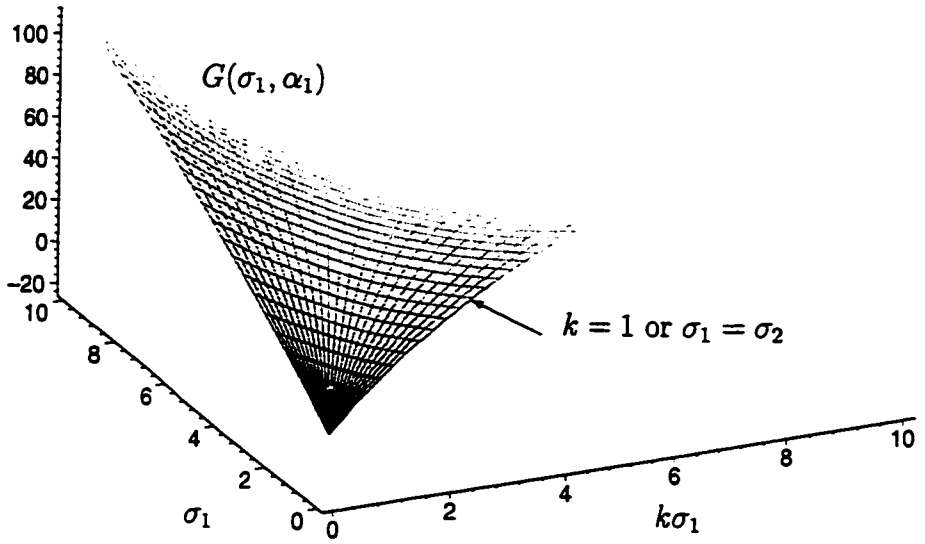


Figure 9.4: Function $G(\sigma_1, k\sigma_1)$ plotted in the ranges $1 \leq \sigma_1 \leq 10$ and $1 \leq k\sigma_1 \leq 10$, where $\sigma_2 = k\sigma_1$.

When it concerns histograms in database query optimization, when k is small, it implies that the optimal and sub-optimal QEPs are very close. Therefore, histogram methods like the Equi-width and the Equi-depth are less likely to find the optimal QEP, since they produce larger errors than histogram approximation methods such as the R-ACM and the T-ACM. The latter produce very small errors, and hence, when comparing any of them with the Equi-width or the Equi-depth, we will have a much larger value of k. This will be reflected in our empirical results presented in the next section.

9.3 Simulation Results for Database Heuristic Functions

9.3.1 Empirical Results

In order to provide practical evidence of the theoretical results presented above[7], we have performed some simulations in database query optimization. In the experiments we have conducted four independent runs. In each run, 100 random databases were generated. Each database was composed of six relations, each of them having six attributes. Each relation was populated with 100 tuples.

For each database, a random query including the six relations and arbitrary attributes was performed. The cost of executing the query using the estimates of the histograms obtained from the Equi-width, the Equi-depth, and the R-ACM was evaluated. This cost is calculated by counting the number of tuples of the intermediate relations involved in the query processing tree. More details of the simulations can be found in [107].

The efficiency of the R-ACM was compared with that of the Equi-width and the Equi-depth after performing these simulations using 50 values per attribute. We set the number of bins for the Equi-width and the Equi-depth to be 22. In order to be impartial with the evaluation, we set the number of bins for the R-ACM to be *approximately half* of that of the Equi-width and the Equi-depth, because the former needs twice as much storage as that of the latter.

The simulation results obtained from 400 independent runs, used to compare the efficiency of the R-ACM with that of the Equi-width and that of the Equi-depth, are given in Table 9.2. The column labeled "R > W" is the number of times that the R-ACM obtains a better solution than that of the Equi-width. The column labeled "W > R" indicates the number of times in which the Equi-width leads to a better QEP than the one determined by the R-ACM. Similarly, the column labeled "R > D"

[7]The empirical results presented in this chapter are not intended to compare the various histogram methods: Equi-width, Equi-depth, R-ACM, T-ACM, V-optimal, etc. The experimental results submitted are merely included to demonstrate that the theoretically proven results can be experimentally justified.

Simulation	R > W	W > R	R > D	D > R
1	26	12	35	12
2	24	15	42	13
3	35	11	46	8
4	29	15	46	8
Total	**114**	**53**	**169**	**41**

Table 9.2: Simulation results for the R-ACM, the Equi-width, and the Equi-depth, after optimizing queries on 400 randomly generated databases. The column labeled "R > W" contains the number of times in which R-ACM obtained a better solution than the Equi-width on 100 randomly generated databases. The information contained in the other columns has a similar interpretation, where "R", "W" and "D" stand for the R-ACM, the Equi-width and the Equi-depth respectively. The last row contains the sum of the values in each column.

represents the number of times that the R-ACM yields a better solution than the Equi-depth, and the column labeled "D > R" is the number of times in which the Equi-depth is superior to the R-ACM. The last row, the total of each column, gives us the evidence that the superiority of the R-ACM over the Equi-width is demonstrated more than twice as often. The same factor relating the superiority of the R-ACM over the Equi-depth is about four.

9.3.2 Geometric Justification of the Rationale

We now present a different perspective for the formulation of the QEP model that has been used earlier. Indeed, we shall analyze the suitability of using the doubly exponential distribution for the query optimization problem. To demonstrate this suitability, we examined 200 randomly selected queries. Since the cost of each query is different for each database, we computed the difference between the actual cost of executing the query and the estimated cost. For each of the histogram methods, namely the Equi-width, the Equi-depth and the R-ACM, we obtained two hundred points[8]. Using these points (or samples) we estimated the parameters of the doubly exponential distribution, λ, for each histogram method, using a Maximum Likelihood

[8]Since these histograms always tend to under-estimate the costs of the queries, we have shifted all the points so that the estimated mean of these samples is zero. In this way, we could work with zero-mean random variables.

Estimate (MLE) method [38].

Given N samples, $\{x_1, \ldots, x_N\}$, obeying a doubly exponential distribution, it is easy (almost purely algebraic) to see that the maximum likelihood parameter, $\hat{\lambda}$, satisfying the distribution obeys:

$$\hat{\lambda} = \frac{N}{\sum_{i=1}^{N} |x_i|}. \tag{9.28}$$

Using the estimate of (9.28), we computed the parameters for the doubly exponential distribution for the Equi-width, the Equi-depth, and the R-ACM, which resulted in 0.6399, 0.6120, and 0.7089 respectively. We have also calculated their variances as in (9.3) – they are 4.8834, 5.3401, and 3.9791 for the Equi-width, the Equi-depth and the R-ACM respectively. As expected, the variance for the R-ACM is smaller than that of the Equi-width and the Equi-depth. This can also be observed in Figure 9.5, in which the corresponding doubly exponential probability distribution functions are plotted for the three histograms. This slight difference between the R-ACM, the Equi-width and the Equi-depth schemes reflects in the corresponding results leading to superior QEPs as shown in Table 9.2. Clearly, the R-ACM, whose variance is smaller than that of the the Equi-width and the Equi-depth, is a superior heuristic function.

In order to observe the similarities between the doubly exponential distribution and the distribution of the actual cost of executing a query, we have plotted the expected values of the doubly exponential distribution and the actual costs obtained when optimizing queries using the R-ACM histogram. The plot depicted in Figure 9.6 was obtained by grouping the data in bins of width two, for the values in the ranges $[x_1, x_2)$, where $x_2 = x_1 + 2$, and $x_2 = 2i$ for $i = -4, \ldots, 5$. In the figure, "R-ACM" (in light gray) represents the actual cost values of the queries, and "d-exp" (in dark gray) represents the expected population in each bin when the random variable is doubly exponential with a value of λ being determined by using (9.28). Observe the similarity between both the histograms. We further corroborate the validity of our model for the database query optimization problem.

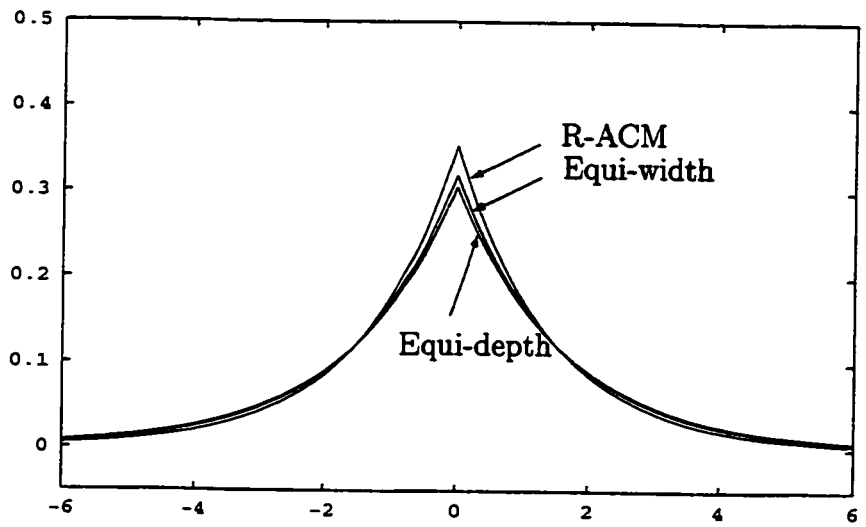


Figure 9.5: Estimated probability density function for three doubly exponential random variables that represent the error in estimation for the Equi-width, the Equi-depth and the R-ACM.

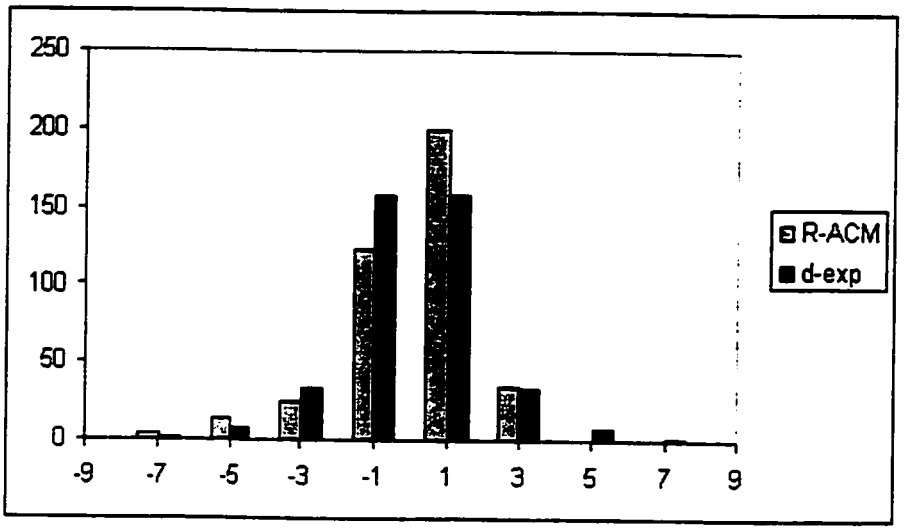


Figure 9.6: Expected values for a doubly exponential random variable, and the actual costs obtained after optimizing queries on 400 random databases using the R-ACM histogram.

9.4 Conclusions

In this chapter, we have applied PR techniques to solve a fundamental open problem in computer science that relates heuristic function accuracy and solution optimality. More specifically, in this chapter, we have discussed the efficiency of using heuristic functions for optimization problems and resolved an open problem, which has been (to our knowledge) open for at least twenty years. The problem involves how the accuracy of a heuristic function relates to the quality of the corresponding solution obtained. The efficiency has been quantified by means of the probability of the heuristic function leading to the optimal solution. We have shown analytically (using a reasonable model of accuracy, namely the doubly exponential distribution for errors) that as the accuracy of a heuristic function increases, the probability of it leading to a superior solution also increases.

Due to the constraints involved in deriving a closed-form expression for integrating the normal probability density function, we have presented a computational analysis of the accuracy/optimality result for the Gaussian distribution. Again, our analysis corroborates the result that heuristic functions producing smaller errors lead more often to optimal solutions.

For the field of database query optimization, we have highlighted that for histogram methods that produce errors with similar variances (the Equi-width and the Equi-depth), the query processing results are also quite similar. However, we have also shown that the R-ACM and the T-ACM, which produce errors with smaller variances than the traditional methods, yield better query optimization plans more often. This result, earlier shown theoretically, has been experimentally verified. Thus, our empirical results on database query optimization show that the R-ACM provides superior solutions more than twice as many times as the Equi-width, and more than four times as often as the Equi-depth. More detailed empirical results including the design of random databases and random queries in these random databases can be found in [107].

We have also estimated the parameters of the doubly exponential distributions representing the Equi-width, the Equi-depth and the R-ACM, and shown graphically how our experiments relate to the theoretical model presented in this chapter.

Part IV

Contributions and Future Directions

Chapter 10

Conclusions

In this thesis, we have presented our contributions to the fields of data compression and pattern recognition. In the area of data compression, we have introduced new approaches and algorithms for the Fano-based static and adaptive coding. We have presented an enhanced version to the static Fano coding, which is more efficient than the traditional Fano coding, and which compresses marginally less than Huffman's algorithm, but uses less memory in the code generation. We have also presented a greedy adaptive version to the Fano coding, which has been found to encode faster than the adaptive Huffman coding, and decode marginally less than the latter. In order to speed up the encoding/decoding phases, we proposed a far more elegant and efficient algorithm to the adaptive Fano coding, which utilizes a new family of trees, namely, the Fano binary search tree.

In the area of pattern recognition, we have formalized the theory for optimal pairwise linear classifiers when the classes are normally distributed. The formal theory has been derived for two dimensional normally distributed random vectors, and also for the multi-dimensional case. In particular, we have re-solved Minsky's paradox for the two-bit parity problem, and for its generalized multidimensional case. We have derived the necessary and sufficient conditions for such classifiers, and empirically tested these classifiers in both synthetic and real-life data sets. We have also derived the formal analysis that relates the accuracy of heuristic functions to the efficiency in searching for optimal (or even suboptimal) solutions to open problems in computer science.

The next few sections discuss an overview of the contributions in each of these cases, both to the fields of data compression and pattern recognition.

10.1 Enhanced Static Fano Coding

In this section, we discuss our contributions, and future potential directions for research for the enhanced static Fano coding problems which we studied in Chapter 3.

10.1.1 Contributions

In Chapter 3, we showed that when the code words are not arranged in an increasing order of length, the coding tree associated with the encoding scheme does not satisfy the sibling property.

In order to rectify this drawback, we presented the enhanced version of Fano's method, namely Fano$^+$, by introducing the concept of "code word rearrangement" into Fano coding. In particular, we formalized the enhanced coding rule, which is a back-end process to the traditional Fano coding, and which incorporates the concept of "code word rearrangement". The chapter also contains the formal encoding and decoding algorithms for Fano$^+$, which includes a recursive procedure that generates the encoding scheme.

The fact that Fano$^+$ generates a prefix code with lossless compression has also been formally proven. Finally, by invoking the code word rearrangement, we have shown that Fano$^+$ is more efficient (or at least as efficient as) the traditional Fano coding.

The experiments for Fano$^+$, the traditional Fano coding and Huffman's algorithm on the files of the Calgary corpus and the Canterbury corpus show that Fano$^+$ is superior to the traditional Fano coding. Fano$^+$ also compresses marginally less than Huffman's algorithm, and uses less memory than the latter during the code generation.

10.1.2 Future Directions

One of the possible extensions of the present work is the development of the corresponding encoding and decoding algorithms for multi-symbol code alphabets.

Another possible future avenue for this work is to propose and study an algorithm that generates a canonical code from the enhanced Fano coding presented here.

Theoretical analysis for the bounds in compression of the enhanced Fano coding, and its comparison with Huffman's algorithm, constitute an interesting problem to be investigated. Finally, another interesting extension of this work constitutes the formalization and (theoretical and empirical) analysis of the enhanced Fano coding for higher-order models, such as Markov models, the LZ family of algorithms, PPM, GBC, etc.

10.2 Greedy Adaptive Fano Coding

In Chapter 4, we introduced the greedy adaptive version to Fano coding. This approach is important in the area of communications and networking, where the *a priori* knowledge of the distributions is not available. We discuss below our contributions in this chapter, and discuss the avenues for future research.

10.2.1 Contributions

In terms of novel contributions, we introduced a greedy approach to the adaptive Fano coding, which typically, generates a sub-optimal encoding scheme, but which uses less memory than the adaptive Huffman coding.

In order to derive the encoding and decoding algorithms, we introduced the concept of *consecutive* partitioning of lists. We proposed the encoding algorithm which processes multi-symbol (typically, ASCII symbols) source alphabets and generates binary code alphabet symbols. The corresponding decoding algorithm for the multi-symbol input and for binary output has also been formalized. These algorithms utilize the principles of Fano coding, namely the nearly-equal-probability property, but proceed by generating a small segment of the entire Fano tree.

We formally showed that the greedy adaptive Fano coding achieves lossless compression. To achieve this, we utilized a doubly-inductive proof on the number of encoding steps and an inner induction on the number of partitioning steps. A corollary to this

proof is that the greedy adaptive Fano coding achieves on-line compression, which is important in many applications, such as the internet, ftp, e-commerce, e-mail, audio and video streaming, etc.

In this chapter, we also derived the time and space complexity of the encoding and decoding algorithms. The worst-case time complexity of the Fano encoding algorithm for the binary output has been found to be linear. Although the space complexity of the greedy adaptive Fano coding is the same as that of the adaptive Huffman coding (i.e. linear), the latter requires about six time as as much space as that of the former. This makes the greedy adaptive Fano suitable for applications in which the memory constraints are tight.

The empirical results conducted on the files of the Calgary corpus and the Canterbury corpus demonstrate that the greedy adaptive Fano coding compresses marginally less than the adaptive Huffman coding. However, the former has been found to be faster in the compression stage, but marginally slower in the decompression phase.

The generalization of the greedy adaptive Fano coding to multi-symbol code alphabets has also been studied in this research work. To help the analysis of the multi-symbol alphabet output scenario, we formally derived an expression for the number of possible consecutive partitionings in terms of the sizes of the source and output alphabets. In order to seek for an optimal solution, we proposed a brute-force algorithm that finds all the possible partitionings for the multi-symbol output alphabet. This algorithm requires polynomial time. However, in order to accelerate the encoding/decoding processes, we also proposed a greedy sub-optimal, but accurate, partitioning procedure. This algorithm operates in linear time.

10.2.2 Future Directions

One of the possible future avenues for the work related to these solutions is to empirical analyze the adaptive greedy Fano encoding and decoding algorithms presented for the multi-symbol code alphabet.

Another open problem consists of developing a non-brute-force algorithm that finds the optimal partitioning for multi-symbol code alphabets. We also believe that it is

possible to formalize the encoding and decoding algorithms for the binary and multi-symbol output versions of the adaptive greedy Fano coding for higher-order models, such as the Markov models, the LZ family of algorithms, PPM, GBC, etc.

Finally, the use of more advanced learning techniques to update the probabilities of the symbols during the encoding process constitutes a future possible avenue of this work. In particular, the use of Learning Automata techniques to update probabilities is promising, and some initial results in this direction are already available.

10.3 Adaptive Fano Coding using Binary Search Trees

Our contributions presented in Chapter 5 relate to the introduction of a novel adaptive approach to Fano coding using a new family of trees, namely the family of Fano binary search trees. In this section, we discuss our related contributions, and present future possible avenues for related research work.

10.3.1 Contributions

In Chapter 5, we introduced a new approach to adaptive Fano coding which utilizes the Fano Binary Search Trees (BSTs). First of all, we defined a new family of trees, namely the partitioning BSTs. This structure is useful in the sense that the source alphabet symbols are incorporated in such a way that their location can be determined using their indices as the keys in the binary search tree. We extended the concept of partitioning BSTs to define a new family of trees, namely the Fano BST, which incorporates the nearly-equal-probability property fundamental to the principles of Fano coding.

In order to maintain the required structure for the Fano BST during the encoding process, we introduced two new tree-based operators, namely, the Shift-To-Left (STL) and the Shift-To-Right (STR) operators. We formalized the rules that implement these two operators for three mutually exclusive cases, and introduced the corresponding algorithms that implement them.

Apart from the two operators mentioned above, we also introduced a heuristic,

called the *conditional shifting heuristic*, which allows us to ensure that the tree maintained during the encoding process satisfies the nearly-equal-probability property. The formal algorithm, which updates the Fano BST during the encoding process and utilizes the conditional shifting heuristic, and the corresponding encoding and decoding algorithms were formalized.

A few relevant properties of the Fano BST coding approach have been proven. In particular, we formally showed that the procedure that updates the Fano BST produces a new tree that also satisfies the properties of a Fano BST. By means of a rigorous, doubly-inductive proof we showed that the encoding and decoding algorithms yield on-line and lossless compression. Finally, we proved that when maintaining a Fano BST using the counter-based updating procedure which we introduced, the adaptive Fano BST asymptotically converges to the Fano tree constructed using the static Fano coding approach. This result is crucial, since it shows that fewer changes are required to the Fano BST as the encoding proceeds, leading to enhanced execution speed.

The speed and compression efficiency of Fano BST coding has been analyzed on empirical data drawn from the well-know benchmarks, namely the Calgary corpus and the Canterbury corpus. Our results show that the adaptive Fano coding using BSTs and the greedy adaptive Fano coding presented in Chapter 4 achieve similar compression ratios. In terms of encoding and decoding speed, the former has been found to be much faster than the latter.

A graphical analysis on the files of the Calgary corpus and the Canterbury corpus, which demonstrates the high performance of the adaptive Fano coding using BSTs was also undertaken. Our plots on the percentage of encoding steps with shift operations, the number of shift operations per node visited, and the estimated probability of invoking shift operations at each encoding step, show that the adaptive BST-based Fano coding asymptotically requires no changes in the Fano BST. This corroborates our theoretical analysis.

10.3.2 Future Directions

One open problem that deserves attention is that of generalizing the result of Theorem 5.5 for *any* other adaptive encoding method that uses an analogous frequency-counter-based updating rule. We believe that similar strategies are applicable for other methods described in Chapter 2, namely Huffman coding, arithmetic coding, Shannon's method, etc.

Another interesting problem to be investigated is the generalization of our adaptive BST-based approach to Fano coding for the case of multi-symbol code alphabets. The possibility of incorporating more advanced learning techniques for updating the Fano BST remains open. Finally the use of higher-order statistical models, such as the Markov model, the LZ family of algorithms, PPM, GBC, etc., in conjunction with our adaptive approach to the Fano coding using BSTs, constitutes an interesting problem to be investigated.

10.4 Optimal Pairwise Linear Classifiers

In Chapters 7 and 8, we presented the formal analysis for the theory of optimal pairwise linear classifiers for two-dimensional normally distributed random vectors. Our contributions and the future possible avenues for research are discussed below.

10.4.1 Contributions

In Chapter 7, we showed that it is possible to find the optimal linear discriminant function for two-dimensional normally distributed random vectors, even when the covariance matrices are different. In most of the cases considered, the functions obtained are *pairs* of straight lines, as opposed to what is known in the previous research, in which the classifier is of the form of a single straight line.

The necessary and sufficient conditions for the optimal pairwise linear classifiers has been derived for three cases. The details of these cases is not re-iterated here. However, we mention the final case where we derived the necessary and sufficient conditions for the optimal pairwise linear classifier when the classes have equal means, and different

covariance matrices, as specified by Minsky's paradox. In each of these cases, the corresponding optimal pairwise linear classifiers were also specified.

The empirical results obtained show that the accuracy of the classifier is very high. The degree of this accuracy is more significant for the case in which are dealing with a linear discriminant function for classes which are significantly overlapping.

Finally, we also presented a scheme from which the maximum likelihood pairwise linear classifier can be estimated. The superiority of this approach over the traditional Fisher's discriminant approach has been experimentally verified on a real life data set, namely the Wisconsin Diagnostic Breast Cancer data set. This superiority has been demonstrated both numerically and graphically.

The two dimensional results of Chapter 7 were extended for the d-dimensional case in Chapter 8. In this case, we formally showed that it is possible to find the optimal linear classifier by satisfying certain conditions specified in the planar projections of the various components. Two other cases, including the one which resolved Minsky's paradox in d dimensions were also studied in detail. Finally the chapter provided some empirical examples for each of the cases mentioned above. The classification accuracy obtained is very high, which is reasonable because the classifier is optimal in the Bayesian sense. The results found in the third case are also impressive given the fact that we are dealing with significantly overlapping classes.

10.4.2 Future Directions

One of the open problems related to our results in this area (for both the two-dimensional and the d-dimensional cases) is to empirically test the optimal pairwise linear classifiers on other real life data sets, and compare their performance with that of other linear classifiers such as Fisher's approach, the minimum squared-error procedures, linear programming algorithms, the perceptron algorithm, etc. But it would be more interesting to theoretically and empirically analyze the classification error for the approximated optimal pairwise linear classifiers. This is a really complex problem since there is no closed-form algebraic expression available for integrating the probability density function of the normal distribution.

Finally, it would be fascinating to study the accuracy of the classifier which utilizes only *one* of the two possible linear classifiers (i.e. the classifier which is based on the most relevant straight line), and to compare the performance of this classifier with the more complete classifier which includes the pair of straight lines.

10.5 A Formal Analysis of Why Heuristic Functions Work

In Chapter 9, we presented a formal analysis for the reason behind why heuristic functions work. The contributions and future directions of our work are discussed below.

10.5.1 Contributions

In order to formalize our model, the efficiency of a heuristic algorithm was quantified by means of the probability of the heuristic function leading to the optimal solution. By modeling the accuracy of the heuristic function probabilistically, we showed that a more accurate heuristic function has a higher probability of leading to a superior solution. These results were verified for various histogram methods used in query optimization. Methods that produce errors with similar variances (the Equi-width and the Equi-depth) yield query processing results which are also quite similar. As opposed to these, the R-ACM and the T-ACM, which produce errors with smaller variances than the traditional methods, have proven to yield better query optimization plans more often.

10.5.2 Future Directions

One interesting problem that can be investigated in this general area is the analysis of heuristics with other probabilistic distributions, such as the one-sided and asymmetric double-sided exponential distributions, the Laplacian distribution, etc. These results could prove relevant for heuristic functions which are not always symmetric.

The application of these to database query optimization problems for non-traditional distributions would also be fascinating.

The study of the accuracy and efficiency of other heuristic algorithms and heuristic functions, such as the A* algorithm, GAs, simulation annealing, hill climbing techniques, etc., in the context of heuristic functions remains entirely open.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Son, 1989.

[2] J. Åberg, Y. M. Shtarkov, and B. J. M. Smeets. Non-Uniform PPM and Context Tree Models. In Storer and Cohn [154], pages 279–288.

[3] N. Abramson. *Information Theory and Coding.* McGraw–Hill, 1963.

[4] G. Adel'son-Vel'skii and E. Landis. An Algorithm for the Organization of Information. *Soviet Math. Dokl.*, 3:1259–1262, 1962.

[5] M. Aladjem. Linear Discriminant Analysis for Two Classes Via Removal of Classification Structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):187–192, 1997.

[6] B. Allen and I. Munro. Self-organizining Binary Search Trees. *J. Assoc. Comput. Mach.*, 25:526–535, 1978.

[7] A. Apostolico and S. Lonardi. Some Theory and Practice of Greedy Off-line - Textual Substitution. In Storer and Cohn [154], pages 119–128.

[8] C. Aragon and R. Seidel. Randomized Search Trees. *Proceedings 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 540–545, 1989.

[9] Z. Arnavut. Move-to-Front and Inversion Coding. In Storer and Cohn [156], pages 193–202.

[10] R. Arnold and T. Bell. A Corpus for the Evaluation of Lossless Compression Algorithms. pages 201–210, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[11] B. Balkenhol, S. Kurtz, and Y. M. Shtarkov. Modifications of the Burrows and Wheeler Data Compression Algorithm. In Storer and Cohn [155], pages 188–197.

[12] P. Barcaccia, A. Cresti, and S. De Agostino. Pattern Matching in Text Compressed with the ID Heuristic. In Storer and Cohn [154], pages 113–118.

[13] T. Bell, J. Cleary, and I. Witten. *Text Compression.* Prentice-Hall, 1990.

[14] S. Bent, D. Sleator, and R. Tarjan. Biased Search Trees. *SIAM Journal of Computing*, 14:545–568, 1985.

[15] J. Bentley and D. McIlroy. Data Compression Using Long Common Strings. In Storer and Cohn [155], pages 287–295.

[16] J. Bitner. Heuristics that Dynamically Organize Data Structures. *SIAM Journal of Computing*, 8:82–110, 1979.

[17] J. Brown and C. Manson. *The Elements of Analiytical Geometry.* McMillan and Co., Limited, 1950.

[18] S. Bunton. *On-Line Stochastic Processes in Data Compression.* PhD thesis, Department of Computer Science and Engineering, University of Washington, March 1997.

[19] S. Bunton. Semantically Motivated Improvements for PPM Variants. *The Computer Journal*, 40(2/3):76–93, 1997.

[20] M. Burrows and D.J. Wheeler. A Block-sorting Lossless Data Compression Algorithm. Technical Report SRC-124, Digital Systems Research Center, Palo Alto, California, May 1994.

[21] B. Chapin. Switching between Two On-line List Update Algorithm for Higher Compression of Burrows-Wheeler Transformed Data. In Storer and Cohn [156], pages 183–192.

[22] B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *Maple V - Reference Manual.* Springer-Verlag, 1991.

[23] R. Cheetham, B.J. Oommen, and D. Ng. Adaptive Structuring of Binary Search Trees Using Conditional Rotations. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):695–704, 1993.

[24] S. Christodoulakis. Estimating selectivities in data bases. In *Technical Report CSRG-136*, Computer Science Dept, University of Toronto, 1981.

[25] C. Chui. *Wavelets: A mathematical Tool for Signal Analysis.* Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[26] J. Cleary and W. Teahan. Unbounded Length Contexts for PPM. *The Computer Journal*, 40(2/3):67–75, 1997.

[27] J. Cleary, W. Teahan, and I. Witten. Unbounded Length Contexts for PPM. pages 52–61, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[28] J. Cleary and I. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[29] C. Cook, A. Rosenfeld, and A. Aronson. Grammatical Inference by Hill Climbing. *Information Science*, 10:59–80, 1976.

[30] G. Cormack and R. Horspool. Algorithms for Adaptive Huffman Codes. *Information Processing Letters*, pages 169–165, 1984.

[31] G. Cormack and R. Horspool. Data Compression Using Dynamic Markov Modelling. *The Computer Journal*, 30(6):541–550, 1987.

[32] T. Cover and J. Thomas. *Elements of Information Theory.* John Wiley & Sons Ltd., 1991.

[33] G. Davis and A. Nosratinia. Wavelet-based Image Coding: An Overview. *Applied and Computational Control, Signals, and Circuits*, 1(1), 1998.

[34] E. Deeba and A. Gunawardena. *Interactive Linear Algebra with MAPLE V.* Springer, 1997.

[35] P. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, 1982.

[36] Q. Du and C. Chang. A Linear Constrained Distance-based Discriminant Analysis for Hyperspectral Image Classification. *Pattern Recognition*, 34(2):361–373, 2001.

[37] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, Inc., 1973.

[38] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* John Wiley and Sons, Inc., New York, NY, 2nd edition, 2000.

[39] S. Dunn. *Visual J++ 6.0 - Developer's Workshop.* Microsoft Press, 1998.

[40] M. Effros. Universal Lossless Source Coding with the Burrows-Wheeler Transform. In Storer and Cohn [155], pages 178–187.

[41] M. Effros. PPM Performance with BWT Complexity: A New Method for Lossless Data Compression. In Storer and Cohn [156], pages 203–212.

[42] P. Elias. Predictive Coding - Part I. *IRE Transactions on Information Theory*, IT-1(1):16–23, 1955.

[43] P. Elias. Predictive Coding - Part II. *IRE Transactions on Information Theory*, IT-1(1):30–33, 1955.

[44] P. Elias. Interval and Recency Rank Source Coding: Two On-Line Adaptive Variable-Length Schemes. *IEEE Transactions on Information Theory*, 33(1):3–10, 1987.

[45] N. Faller. An Adaptive System for Data Compression. *Seventh Asilomar Conference on Circuits, Systems, and Computers*, pages 593–597, 1973.

[46] M. Feder and A. C. Singer. Universal Data Compression and Linear Prediction. In Storer and Cohn [154], pages 511–520.

[47] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[48] R. Gallager. Variations on a Theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[49] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W H Freeman & Co., 1979.

[50] L. Gąsieniec and W. Rytter. Almost Optimal Fully LZW-compressed Pattern Matching. In Storer and Cohn [155], pages 317–326.

[51] J. Gibson. Adaptive Prediction in Speech Differential Encoding Systems. *Proceedings of IEEE*, 68:488–525, April 1980.

[52] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Pub, 1997.

[53] R. Gray. *Entropy and Information Theory.* Springer-Verlag, 1990.

[54] T. M. Ha. The Optimum Class-Selective Rejection Rule. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):608–615, 1997.

[55] T. M. Ha. Optimum Decision Rules in Pattern Recognition. *Advances in Pattern Recognition, SSPR'98 - SPR'98*, pages 726–735, 1998.

[56] R. Hamming. *Coding and Information Theory.* Prentice Hall, 1980.

[57] R. Hamming. *Coding and Information Theory.* Prentice Hall, 2nd. edition, 1986.

[58] D. Hankerson, G. Harris, and P. Johnson Jr. *Introduction to Information Theory and Data Compression.* CRC Press, 1998.

[59] G. Hardy, J. Littlewood, and G. Póyla. *Inequalities.* Cambridge University Press, 2nd. edition, 1959.

[60] G. Held. *Data and Image Compression: Tools and Techniques.* John Wiley & Sons, 1996.

[61] D. Hirschberg and D. Lelewer. Efficient Decoding of Prefix Codes. *Communications of the ACM*, 33(4):464–473, 1990.

[62] D. Hoffman, D. Leonard, C. Lindner, K. Phelps, C. Rodger, and J. Wall. *Coding Theory: The Essentials.* Marcel Dekker, 1992.

[63] R. Hoffman. *Data Compression in Digital Systems.* Chapman & Hall, 1996.

[64] P. Howard. The Design and Analysis of Efficient Lossless Data Compression Systems. Technical Report CS-93-28, School of Computer Science, Brown University, Providence, Rhode Island, June 1993.

[65] P. Howard and J. Vitter. Analysis of Arithmetic Coding for Data Compression. In Storer and Reif [157], pages 3–12.

[66] D.A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of IRE*, 40(9):1098–1101, 1952.

[67] Kevin Ingalls and Daniel Jinguji. *Learn Microsoft Visual J++ 6.0 Now.* Microsoft Press, 1998.

[68] Y. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of the ACM-SIGMOD Conference*, pages 268–277, 1991.

[69] N. Jayant and P. Noll. *Digital Coding of Waveforms.* Englewood Cliffs, NJ: Prentice Hall, 1984.

[70] S. Karlin and H. Taylor. *A First Course in Stochastic Processes.* Academic Press, 2nd. edition, 1974.

[71] T. Kida, M. Takeda, and A. Shinohara. Multiple Pattern Matching in LZW Compressed Text. In Storer and Cohn [154], pages 103–112.

[72] J. C. Kieffer and E. Yang. Grammar-Bassed Codes: A new Class of Universal Lossless Source Codes. *IEEE Transactions on Information Theory*, 46(3):737–754, May 2000.

[73] J. C. Kieffer, E. Yang, G. Nelson, and P. Cosman. Universal Lossless Compression via Multilevel Pattern Matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, July 2000.

[74] D. Knuth. *The Art of Computer Programming*, volume 3. Reading, MA: Addison-Wesley, 1973.

[75] D. Knuth. Dynamic Huffman Coding. *Journal of Algorithms*, 6:163–180, 1985.

[76] R. P. Kooi. *The optimization of queries in relational databases.* PhD thesis, Case Western Reserve University, 1980.

[77] D. Kreher and D. Stinson. *Combinatorial Algorithms : Generation, Enumeration, and Search.* CRC Press, 1998.

[78] W. Krzanowski, P. Jonathan, W. McCarthy, and M. Thomas. Discriminant Analysis with Singular Covariance Matrices: Methods and Applications to Spectroscopic Data. *Applied Statistics*, 44:101–115, 1995.

[79] T. Lai and D. Wood. Adaptive Heuristics for Binary Search Trees and Constant Linkage Cost. *SIAM Journal of Computing*, 27(6):1564–1591, December 1998.

[80] J. Michael Lake. Prediction by Grammatical Match. In Storer and Cohn [156], pages 153–162.

[81] G. Langdon. An Introduction to Arithmetic Coding. *IBM Journal of Research and Development*, 18(2):135–149, 1984.

[82] G. Langdon and J. Rissanen. Compression of Black-white Images with Arithmetic Coding. *IEEE Transactions on Communications*, 29(6):858–867, 1981.

[83] N. J. Larsson and A. Moffat. Offline Dictionary-Based Compression. In Storer and Cohn [155], pages 296–305.

[84] C. Lau, editor. *Neural Networks: Theoretical Foundations and Analysis.* IEEE Press, 1992.

[85] H. Lekatsas and W. Wolf. Random Access Decompression using Binary Arithmetic Coding. In Storer and Cohn [155], pages 306–315.

[86] D. Levy. *How Computers Play Chess.* Computer Science Press, New York, 1991.

[87] R. Lippman. An Introduction to Computing with Neural Nets. In Lau [84], pages 5–24.

[88] R. Lotlikar and R. Kothari. Adaptive Linear Dinensionality Reduction for Classification. *Pattern Recognition*, 33(2):185–194, 2000.

[89] W. Malina. On an Extended Fisher Criterion for Feature Selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:611–614, 1981.

[90] O. Mangasarian, W. Street, and W. Wolberg. Breast Cancer Diagnosis and Prognosis via Linear Programming. *Operations Research*, 43(4):570–577, 1995.

[91] M.V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. In *ACM Computing Surveys*, volume 20, pages 192–221, 1988.

[92] P. Maragos, R. Schafer, and R Mersereau. Two Dimensional Linear Prediction and Its Application to Adaptive Predictive Coding of Images. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, ASSP-32:1213–1229, December 1984.

[93] Y. Matias and S.C. Sahinalp. On Optimality of Parsing in Dynamic Dictonary Based Data Compression. *ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[94] K. Mehlhorn. Dynamic Binary Search. *SIAM Journal of Computing*, 8:175–198, 1979.

[95] M. Michell. *An Introduction to Genetic Algorithms* . MIT Press, 1998.

[96] M. Minsky. *Perceptrons.* MIT Press, 2nd edition, 1988.

[97] A. Moffat. Implementing The PPM Data Compression Scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[98] A. Moffat, N. Sharman, and I. Witten. An Empirical Evaluation of Coding Methods for Multi-symbol Alphabet. *Information Processing & Management*, 30(6):791–804, 1994.

[99] A. Moffat and A. Turpin. Efficient Construction of Minimum-redundancy Codes for Large Alphabets. *IEEE Transactions on Information Theory*, 44(4):1650–1657, July 1998.

[100] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of ACM-SIGMOD Conference*, pages 28–36, 1988.

[101] O. Murphy. Nearest Neighbor Pattern Classification Perceptrons. In Lau [84], pages 263–266.

[102] Kumpati Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[103] C. Nevill-Manning and I. Witten. Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm. *Journal of Artificial Intellingence Research*, 7:67–82, 1997.

[104] N. Nilsson. *Artificial Intelligence : A New Synthesis.* Morgan Kaufmann Publishers, 1998.

[105] B. J. Oommen and L. Rueda. The Efficiency of Modern-day Histogram-like Techniques for Query Optimization. To appear in *The Computer Journal.*

[106] B. J. Oommen and L. Rueda. Histogram Methods in Query Optimization: The Relation between Accuracy and Optimality. In *Proceedings of the 7th International Conference on Database Systems for Advanced Applications*, pages 320–326, Hong Kong, April 2001.

[107] B. J. Oommen and L. Rueda. The Efficiency of Modern-day Histogram-like Techniques for Query Optimization. Technical Report TR-01-06, School of Computer Science, Carleton University, Ottawa, Canada, June 2001.

[108] B. J. Oommen and L. Rueda. Using Pattern Recognition Techniques to Derive a Formal Analysis of Why Heuristic Functions Work. In *Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems*, pages 45–58, Alicante, Spain, April 2002.

[109] B. J. Oommen and M. Thiyagarajah. The Rectangular Attribute Cardinality Map: A New Histogram-like Technique for Query Optimization. In *Proceedings of the International Database Engineering and Applications Symposium, IDEAS'99*, pages 3–15, Montreal, Canada, August 1999.

[110] B. J. Oommen and M. Thiyagarajah. On the Use of the Trapezoidal Attribute Cardinality Map for Query Result Size Estimation. In *Proceedings of the 2000 International Database Engineering and Applications Symposium*, pages 236–242, Yokohama, Japan, September 2000.

[111] B.J. Oommen and T. De St. Croix. Graph Partitioning Using Learning Automata. *IEEE Transactions on Computers*, 45(2):195–208, 1995.

[112] B.J. Oommen and L. Rueda. A Formal Analysis of Why Heuristic Functions Work. To appear in *Artificial Intelligence*.

[113] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw–Hill, 3rd. edition, 1991.

[114] W. Pennebaker and J. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1992.

[115] F. Peracchi. *Econometrics*. John Wiley & Sons, 2001.

[116] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of ACM-SIGMOD Conference*, pages 256–276, 1984.

[117] V. Pless. *Introduction to the Theory of Error-Correcting Codes.* John Wiley & Sons, 3rd. edition, 1998.

[118] W. Poosala. *Histogram Based Estimation Techniques in Databases.* PhD thesis, University of Wisconsin - Madison, 1997.

[119] A. Rao, D. Miller, K. Rose, , and A. Gersho. A Deterministic Annealing Approach for Parsimonious Design of Piecewise Regression Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(2):159–173, 1999.

[120] K. Rao and R. Yip. *Discrete Cosine Transform: Algorithms, Advantages, and Applications.* Academic Press, 1990.

[121] S. Raudys. On Dimensionality, Sample Size, and Classification Error of Nonparametric Linear Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):667–671, 1997.

[122] S. Raudys. Evolution and Generalization of a Single Neurone: I. Single-layer Perception as Seven Statistical Classifiers. *Neural Networks*, 11(2):283–296, 1998.

[123] S. Raudys. Evolution and Generalization of a Single Neurone: II. Complexity of Statistical Classifiers and Sample Size Considerations. *Neural Networks*, 11(2):297–313, 1998.

[124] E. Rich and K. Knight. *Artificial Intelligence.* McGraw Hill, 2nd edition, 1991.

[125] B. Ripley. *Pattern Recognition and Neural Networks.* Cambridge Univ. Press, 1996.

[126] Jorma Rissanen. Fast Universal Coding with Context Models. *IEEE Transactions on Information Theory*, 45(4):1065–1071, May 1999.

[127] S. Roman. *Coding and Information Theory.* Springer-Verlag, 1992.

[128] G. Romp. *Game Theory: Introduction & Applications.* Oxford University Press, 1997.

[129] K. Rosen. *Discrete Mathematics and Its Applications.* McGraw–Hill Canada, 4th. edition, 1998.

[130] L. Rueda and B. J. Oommen. Greedy Adaptive Fano Coding for Binary and Multi-Symbol Code Alphabets. (Submited for Publication).

[131] L. Rueda and B. J. Oommen. Nearly Optimal Fano-Based Canonical Codes. (Submited for Publication). Also available as a Technical Report TR-01-05, School of Computer Science, Carleton University, Ottawa, Canada, 2001.

[132] L. Rueda and B. J. Oommen. Enhanced Static Fano Coding. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, pages 2163–2169, Tucson, Arizona, USA, 2001.

[133] L. Rueda and B. J. Oommen. Greedy Adaptive Fano Coding. In *Proceedings of the 2002 IEEE Aerospace Conference*, BigSky, MT, USA. Track 10.0407, March 2002.

[134] Luis G. Rueda and B. John Oommen. On Optimal Pairwise Linear Classifiers for Normal Distributions: The d-Dimensional Case. Technical Report SCS-TR-01-03, School of Computer Science, Carleton University, Ottawa, Canada.

[135] Luis G. Rueda and B. John Oommen. On Optimal Pairwise Linear Classifiers for Normal Distributions: The d-Dimensional Case. To appear in *Pattern Recognition.*

[136] Luis G. Rueda and B. John Oommen. On Optimal Pairwise Linear Classifiers for Normal Distributions: The Two-Dimensional Case. Technical Report TR-00-03, School of Computer Science, Carleton University, Ottawa, Canada, May 2000.

[137] Luis G. Rueda and B. John Oommen. The Foundational Theory of Optimal Bayesian Pairwise Linear Classifiers. In *Proceedings of the Joint IAPR International Workshops SSPR 2000 and SPR 2000*, pages 581–590, Alicante, Spain, August/September 2000. Springer.

[138] Luis G. Rueda and B. John Oommen. Resolving Minsky's Paradox: The *d*-Dimensional Normal Distribution Case. In *The 14th Australian Joint Conference on Artificial Intelligence (AI'01)*, pages 25–36, Adelaide, Australia, December 2001.

[139] Luis G. Rueda and B. John Oommen. On Optimal Pairwise Linear Classifiers for Normal Distributions: The Two-Dimensional Case. *IEEE Transations on Pattern Analysis and Machine Intelligence*, 24(2):274–280, 2002.

[140] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice–Hall, New York, 1995.

[141] K. Sadakane. A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation. In Storer and Cohn [154], pages 129–138.

[142] K. Sadakane, T. Okazaki, and H. Imai. Implementing the Context Tree Wighting Method for Text Compression. In Storer and Cohn [156], pages 123–132.

[143] S. Savari. Redundancy of the Lempel–Ziv Incremental Parsing Rule. *IEEE Transactions on Information Theory*, 43(1):9–21, 1997.

[144] W. Savitch. *Java: An Introduction to Computer Science Programming.* Prentice Hall, 1999.

[145] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann, 2nd. edition, 2000.

[146] R. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley and Sons, Inc., 1992.

[147] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communications.* University of Illinois Press, 1949.

[148] M. Sherk. Self-adjusting k-ary Search Trees and Self-adjusting Balanced Search Trees. Technical Report 234/90, University of Toronto, Toronto, Canada, February 1990.

[149] Paul C. Shields. Performance of LZ Algorithms on Individual Sequences. *IEEE Transactions on Information Theory*, 45(4):1283–1288, May 1999.

[150] D. Sleator and R. Tarjan. Self-adjusting Binary Search Trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.

[151] T. Smith and R. Peeters. Fast Convergence with a Greedy Tag-Phrase Dictionary. In Storer and Cohn [154], pages 33–42.

[152] J. Storer. *Data Compression: Methods and Theory.* Computer Science Press, 1988.

[153] J. Storer and T. Szymanski. Data Compression via Textual Substitution. *Journal of the ACM*, 29(9):928–951, 1982.

[154] James A. Storer and Martin Cohn, editors. *Proceedings, Data Compression Conference*, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[155] James A. Storer and Martin Cohn, editors. *Proceedings, Data Compression Conference*, Los Alamitos, CA, 1999. IEEE Computer Society Press.

[156] James A. Storer and Martin Cohn, editors. *Proceedings, Data Compression Conference*, Los Alamitos, CA, 2000. IEEE Computer Society Press.

[157] James A. Storer and John H. Reif, editors. *Proceedings, Data Compression Conference*, Los Alamitos, CA, 1991. IEEE Computer Society Press.

[158] L. Stuiver and A. Moffat. Piecewise Integer Mapping for Arithmetic Coding. In Storer and Cohn [154], pages 3–12.

[159] I. Tăbuş, G. Kóródi, and J. Rissanen. Text Compression Based on Variable-to-Fixed Codes for Markov Sources. In Storer and Cohn [156], pages 133–141.

[160] P. A. Volf and F. M. Williems. Switching between Two Universal Source Coding Algorithms. In Storer and Cohn [154], pages 491–500.

[161] W. Walker and C. Gotlieb. A Top-down Algorithm for Constructing Nearly Optimal Lexicographical Trees. In Academic Press, editor, *Graph Theory and Computing*, New York, 1972.

[162] A. Webb. *Statistical Pattern Recognition.* Oxford University Press Inc., New York, 1999.

[163] M. J. Weinberger and E. Ordentlich. On-line Decision Making for a Class of Loss Function via Lempel-Ziv Parsing. In Storer and Cohn [156], pages 163–172.

[164] T. Welch. A technique for High Performance Data Compression. *IEEE Computer*, 16(6):8–19, 1984.

[165] F. M. Willems. The Context-Tree Weighting Method: Extensions. *IEEE Transactions on Information Theory*, 44:792–798, March 1998.

[166] F. M. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The Context-Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory*, 41:653–664, May 1995.

[167] R. Williams. *Adaptive Data Compression.* Kluwer Academic Publishers, 1991.

[168] R. Williams. An Extremely Fast Ziv-Lempel Data Compression Algorithm. In Storer and Reif [157], pages 8–11.

[169] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann, 2nd. edition, 1999.

[170] E. Yang and J. C. Kieffer. Efficient Universal Lossless Data Compression Algorithms Based on a Greedy Sequential Grammar Transform–Part One: Without Context Models. *IEEE Transactions on Information Theory*, 46(3):755–777, May 2000.

[171] H. Yokoo. Context Tables: A Tool for Describing Text Compression Algorithms. In Storer and Cohn [154], pages 299–308.

[172] A. Yuille and M. Coughlan. An A* Perspective on Deterministic Optimization for Deformable Templates. *Pattern Recognition*, 33:603–616, 2000.

[173] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[174] J. Ziv and A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory*, 25(5):530–536, 1978.